

# PySpark Cheatsheet

---

## 1. PySpark Overview

- **Definition:** PySpark is the Python API for Apache Spark, an open-source, distributed computing framework.
  - **Core Components:**
    - **RDD (Resilient Distributed Dataset):** Immutable distributed collections of objects.
    - **DataFrame:** Distributed table with named columns; optimized for SQL queries.
    - **Dataset:** Strongly typed, distributed data structure (available in Scala/Java).
  - **Languages Supported:** Python, Scala, Java, and R.
- 

## 2. Core Spark Concepts

- **Driver:** Manages the execution of tasks across the cluster.
  - **Executor:** Performs computations and stores data on worker nodes.
  - **Partition:** Logical division of data for parallel processing.
  - **Transformations:** Create a new RDD/DataFrame from an existing one (e.g., map, filter).
  - **Actions:** Trigger execution of transformations and return results (e.g., count, collect).
- 

## 3. PySpark Architecture

- **Cluster Manager:**
  - YARN, Mesos, or Standalone cluster.
- **Execution Process:**
  1. Job submitted by Driver.
  2. Tasks divided into Stages based on shuffle boundaries.
  3. Tasks run on Executors.

---

## 4. Common PySpark Operations

- **Transformations:**
  - **map:** Applies a function to each element.
  - **filter:** Filters elements based on a condition.
  - **groupBy:** Groups data by a key.
  - **join:** Joins two DataFrames based on a condition.
- **Actions:**
  - **show:** Displays DataFrame.
  - **collect:** Brings data to the driver.
  - **count:** Counts the number of elements.

---

## 5. PySpark SQL

- **Creating a Table:**
- `df.createOrReplaceTempView("table_name")`
- `spark.sql("SELECT * FROM table_name")`
- **Common SQL Functions:**
  - **agg:** Perform aggregations.
  - **alias:** Rename columns.
  - **distinct:** Remove duplicates.

---

## 6. Window Functions

- **Definition:** Perform operations over a window of rows.
- **Types:**
  - **Ranking:** `row_number`, `rank`, `dense_rank`.
  - **Aggregations:** `sum`, `avg`, `max`, `min`.
- **Example:**
- `from pyspark.sql.window import Window`

- `window_spec = Window.partitionBy("col1").orderBy("col2")`
  - `df.withColumn("rank", rank().over(window_spec))`
- 

## 7. DataFrame API vs. SQL API

- **DataFrame API:**
    - Pythonic syntax.
    - Example: `df.select("col1", "col2").filter(df["col3"] > 10)`
  - **SQL API:**
    - SQL-like syntax.
    - Example: `spark.sql("SELECT col1, col2 FROM table WHERE col3 > 10")`
- 

## 8. Persisting and Caching

- **Caching:** Stores data in memory for faster reuse.
  - `df.cache()`
  - **Persistence:** Allows control over storage levels (e.g., `MEMORY_AND_DISK`).
  - `df.persist(StorageLevel.DISK_ONLY)`
- 

## 9. Joins in PySpark

- **Types of Joins:**
    - Inner, Left, Right, Full Outer, Semi, Anti.
  - **Broadcast Join:** Optimized join when one DataFrame is small.
  - `from pyspark.sql.functions import broadcast`
  - `df = large_df.join(broadcast(small_df), "key")`
- 

## 10. File Formats

- **Supported Formats:** CSV, JSON, Parquet, Avro, ORC.
- **Reading Data:**
- `df = spark.read.format("csv").option("header", True).load("path")`

- **Writing Data:**
  - `df.write.format("parquet").save("path")`
- 

## 11. Performance Optimization

- **Repartitioning:** Adjust the number of partitions for parallelism.
  - `df.repartition(10)`
  - **Coalesce:** Reduce the number of partitions without a shuffle.
  - `df.coalesce(1)`
  - **Predicate Pushdown:** Filters data early in the query execution.
- 

## 12. Streaming with PySpark

- **Reading Streams:**
  - `df = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").load()`
  - **Writing Streams:**
  - `query = df.writeStream.format("console").start()`
  - `query.awaitTermination()`
- 

## 13. Error Handling

- **Common Exceptions:**
    - **AnalysisException:** Invalid query or missing columns.
    - **Py4JJavaError:** Java exception in Spark operations.
  - **Debugging:**
    - Use `.explain()` to understand the query execution plan.
    - Check Spark logs for detailed error messages.
- 

## 14. Common PySpark Interview Questions

- What are the differences between RDD, DataFrame, and Dataset?

- **How does Spark handle fault tolerance?**
  - **Explain the concept of lazy evaluation in PySpark.**
  - **How do you optimize joins in PySpark?**
  - **Explain Spark's execution process (job, stages, and tasks).**
- 

## **1. PySpark Basics**

### **Initialize SparkSession:**

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("AppName").getOrCreate()
```

### **Create DataFrame:**

```
data = [(1, "Alice"), (2, "Bob")]

columns = ["id", "name"]

df = spark.createDataFrame(data, columns)
```

### **Inspect DataFrame:**

```
df.show()    # Display rows

df.printSchema() # Show schema

df.describe().show() # Summary statistics
```

### **Read/Write Data:**

```
# Read

df = spark.read.csv("file_path", header=True, inferSchema=True)

# Write

df.write.csv("output_path", header=True)
```

---

## **2. PySpark SQL**

### **SQL Queries:**

```
df.createOrReplaceTempView("table")
```

```
spark.sql("SELECT * FROM table WHERE id > 1").show()
```

### **Joins:**

```
df1.join(df2, df1["key"] == df2["key"], "inner").show() # Types: inner, left, right, outer
```

---

## **3. Transformations**

### **Basic Transformations:**

```
df.select("column1", "column2").show() # Select columns
```

```
df.filter(df["column"] > 10).show() # Filter rows
```

```
df.withColumn("new_col", df["col"] * 2).show() # Add column
```

### **GroupBy and Aggregations:**

```
from pyspark.sql.functions import count, avg, sum
```

```
df.groupBy("column").agg(count("*").alias("count"), avg("col2")).show()
```

### **Window Functions:**

```
from pyspark.sql.window import Window
```

```
from pyspark.sql.functions import rank
```

```
window = Window.partitionBy("category").orderBy("sales")
```

```
df.withColumn("rank", rank().over(window)).show()
```

---

## **4. PySpark Functions**

### **Common Functions:**

```
from pyspark.sql.functions import col, lit, concat, when
```

```
df = df.withColumn("new_col", concat(col("col1"), lit("_"), col("col2"))) # Concatenate
```

```
df = df.withColumn("status", when(df["col"] > 10, "High").otherwise("Low")) #  
Conditional
```

### **Date Functions:**

```
from pyspark.sql.functions import current_date, datediff
```

```
df = df.withColumn("today", current_date())
```

```
df = df.withColumn("days_diff", datediff(df["date_col"], df["today"]))
```

---

## 5. PySpark RDD Operations

### Basic RDD Operations:

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4])
```

```
mapped_rdd = rdd.map(lambda x: x * 2)
```

```
filtered_rdd = mapped_rdd.filter(lambda x: x > 4)
```

```
print(filtered_rdd.collect())
```

### Actions:

```
print(rdd.count())
```

```
print(rdd.collect())
```

### Transformations:

```
rdd1 = spark.sparkContext.parallelize([1, 2, 3])
```

```
rdd2 = spark.sparkContext.parallelize([3, 4, 5])
```

```
union_rdd = rdd1.union(rdd2)
```

```
intersection_rdd = rdd1.intersection(rdd2)
```

---

## 6. PySpark Optimization

### Persist and Cache:

```
df.cache()    # Cache in memory
```

```
df.persist()  # Persist in memory and disk
```

```
df.unpersist() # Remove from cache
```

### Repartition:

```
df = df.repartition(4) # Increase partitions
```

```
df = df.coalesce(2)    # Decrease partitions
```

---

## 7. PySpark Interview Patterns

### 1. Self Join Example:

```
df.alias("df1").join(df.alias("df2"), col("df1.id") == col("df2.supervisor"), "inner").show()
```

### 2. Window Function Example:

```
from pyspark.sql.functions import row_number
```

```
window = Window.partitionBy("category").orderBy("sales")
```

```
df.withColumn("row_number", row_number().over(window)).show()
```

### 3. Aggregate Example:

```
df.groupBy("department").agg(  
    count("*").alias("count"),  
    avg("salary").alias("avg_salary")  
)
```

---

## 8. PySpark Advanced Topics

### Broadcast Joins:

```
from pyspark.sql.functions import broadcast
```

```
df_large.join(broadcast(df_small), "key").show()
```

### UDF (User-Defined Functions):

```
from pyspark.sql.functions import udf
```

```
from pyspark.sql.types import StringType
```

```
def uppercase(name):
```

```
    return name.upper()
```

```
uppercase_udf = udf(uppercase, StringType())
```



```
df.withColumn("uppercase_name", uppercase_udf(df["name"])).show()
```

### **Accumulators:**

```
acc = spark.sparkContext.accumulator(0)
```

```
def add_to_acc(value):
```

```
    acc.add(value)
```

```
rdd.foreach(add_to_acc)
```

```
print(acc.value)
```

---

## **11. Broadcast Joins**

- **Definition:** Optimizes join operations when one DataFrame is small enough to fit in memory.
- **Syntax:**

```
from pyspark.sql.functions import broadcast
```

```
result = large_df.join(broadcast(small_df), "key")
```

- **Use Case:** Useful for improving performance by avoiding shuffle operations.

## **12. Window Functions**

- **Usage:** Perform operations like ranking, cumulative sums, etc., over a specific window of rows.
- **Example:**

```
from pyspark.sql.window import Window
```

```
from pyspark.sql.functions import rank, col
```

```
window_spec = Window.partitionBy("department").orderBy("salary")
```

```
ranked_df = employees.withColumn("rank", rank().over(window_spec))
ranked_df.show()
```

- **Common Functions:** row\_number, rank, dense\_rank, lag, lead, ntile.

### 13. Data Partitioning

- **Repartitioning:** Changes the number of partitions.

```
df_repartitioned = df.repartition(4)
```

- **Coalesce:** Reduces the number of partitions without shuffling.

```
df_coalesced = df.coalesce(2)
```

### 14. Accumulators

- **Definition:** Variables used to perform aggregations.
- **Syntax:**

```
acc = spark.sparkContext.accumulator(0)
rdd.foreach(lambda x: acc.add(1))
print(acc.value)
```

### 15. Caching and Persistence

- **Caching:** Stores RDD/DataFrame in memory for reuse.

```
df.cache()
```

- **Persistence:** Allows specifying storage levels (e.g., memory, disk).

```
df.persist(StorageLevel.MEMORY_AND_DISK)
```

## 16. Skew Handling

- **Salting:** Add random prefixes to keys to distribute data evenly during joins.
- **Example:**

```
df_with_salt = df.withColumn("salted_key", concat(col("key"), lit("_"), col("random_id")))
```

## 17. Fault Tolerance

- **RDD Lineage:** RDDs keep track of transformations for automatic recovery in case of failure.
- **Action Retry:** Automatically retries failed tasks.

## 18. Integration with Other Tools

- **Integration with Hive:**

```
spark.sql("SELECT * FROM hive_table")
```

- **Reading/Writing to Kafka:**

```
df = spark.readStream.format("kafka").option("kafka.bootstrap.servers",  
"localhost:9092").option("subscribe", "topic1").load()
```

```
df.writeStream.format("kafka").option("kafka.bootstrap.servers",  
"localhost:9092").option("topic", "topic2").start()
```

## 19. Advanced File Formats

- **Avro:**

```
df.write.format("avro").save("path")
```

- **ORC:**

```
df.write.format("orc").save("path")
```

## 20. Performance Tuning

- **Common Parameters:**

- spark.sql.shuffle.partitions: Adjust for better parallelism.
- spark.executor.memory: Increase memory for executors.
- spark.executor.cores: Set the number of cores per executor.

- **Example:**

```
spark.conf.set("spark.sql.shuffle.partitions", 50)
```