**Here are few Pyspark components used Databricks workspace with concise definitions and examples**

1. **RDDs**: Resilient Distributed Datasets, the fundamental data structure in PySpark for distributed data.
2. **Data Frames**: Distributed tables with named columns, similar to a table in SQL.

Example: df.filter(df.age > 21).

1. **Transformations**: Operations that create a new RDD/DataFrame, like map() or filter(). Example: rdd.map(lambda x: x\*2).
2. **Actions**: Operations that trigger computations, like collect() or count().

Example: rdd.count().

1. **Spark Session**: The entry point to programming Spark with DataFrames.

Example: spark = SparkSession.builder.appName("app").getOrCreate().

1. **Spark Context**: The primary entry point for Spark functionality.

Example: sc = SparkContext().

1. **Data Source APIs**: Allow reading/writing data in various formats (JSON, CSV, etc.). Example: spark.read.json("path").
2. **Partitions**: Logical divisions of data for parallel processing.

Example: rdd.getNumPartitions().

1. **PySpark SQL**: Module for working with structured data using SQL.

Example: df.createOrReplaceTempView("table"); spark.sql("SELECT \* FROM table").

1. **UDFs (User-Defined Functions)**: Custom functions for DataFrames.

Example: from pyspark.sql.functions import udf.

1. **Joins**: Combining DataFrames based on a common column.

Example: df1.join(df2, "id").

1. **Broadcast Variables**: Efficiently share large read-only data across nodes.

Example: sc.broadcast(value).

1. **Accumulators**: Variables used for aggregating information across the cluster.

Example: accum = sc.accumulator(0).

1. **Caching**: Storing intermediate results for reusing.

Example: df.cache().

1. **PySpark Streaming**: Real-time data processing.

Example: df.writeStream.format("console").start().

1. **MLlib**: Spark’s machine learning library.

Example: from pyspark.ml.classification import LogisticRegression.

1. **Cluster Manager**: Manages resources across applications (e.g., YARN, Mesos).
2. **Tasks**: Units of work sent to executors.
3. **Executors**: Run tasks on worker nodes.
4. **SQL**: Integration with SQL queries. Example: spark.sql("SELECT \* FROM table").
5. **Overwrite**: Mode for saving data that replaces existing data.

Example: df.write.mode("overwrite").save("path").

**22. Append**: Mode for saving data that adds new data to existing data.

Example: df.write.mode("append").save("path").

1. **Pandas**: A Python library for data manipulation and analysis.

Example: import pandas as pd; df = pd.read\_csv("file.csv").

1. **pandas\_udf**: User-defined function using Pandas for vectorized operations.

Example: from pyspark.sql.functions import pandas\_udf.

1. **PySpark Streaming**: Processing real-time data streams.

Example: df.writeStream.format("console").start().

1. To read data from a Data Lake, you can use **DataFrameReader**. Specifically, the spark.read method allows you to load data from various sources, including Azure Data Lake Storage.

Example: df = spark.read.format("parquet").load("path\_to\_datalake")

**Here are 15 Python concepts related to Data Engineering, PySpark, and Databricks, with concise definitions and examples:**

1. **Lambda Functions**:  
   Anonymous, inline functions defined with lambda.  
   Example: squared = lambda x: x\*\*2.
2. **List Comprehensions**:  
   Compact way to generate lists from iterables.  
   Example: [x\*\*2 for x in range(5)].
3. **Decorators**:  
   Functions modifying another function’s behavior.  
   Example: @log\_time.
4. **Generators**:  
   Yield-based iterators for memory efficiency.  
   Example: def gen(): yield x.
5. **Exception Handling**:  
   Managing runtime errors with try-except.  
   Example: try: 1/0 except: pass.
6. **Modules and Packages**:  
   Organized code in reusable libraries.  
   Example: import os.
7. **Pandas DataFrames**:  
   Tabular data manipulation library.  
   Example: pd.DataFrame(data).
8. **pyspark.sql**:  
   Spark SQL integration for structured data.  
   Example: spark.read.csv(path).
9. **Broadcast Variables**:  
   Share large read-only data efficiently.  
   Example: sc.broadcast(value).
10. **Accumulators**:  
    Variables for aggregated metrics.  
    Example: accum.add(1).
11. **Context Managers**:  
    Manage resources (e.g., files) with with.  
    Example: with open() as file.
12. **Classes and Objects**:  
    Create reusable blueprints (OOP).  
    Example: class DataPipeline:.
13. **File Handling**:  
    Read/write files with Python's open().  
    Example: f.write(data).
14. **Datetime Library**:  
    Handle timestamps and dates.  
    Example: datetime.now().
15. **APIs (Requests)**:  
    Interact with external services using requests.  
    Example: requests.get(url).
16. **Lists**: Ordered collections of elements, supporting duplicates

**Use in PySpark**: Commonly used for collecting column names, values, or building data for RDDs

Example: columns = ["id", "name", "age"] print(columns[1]) # Output: name

1. **Dictionaries**: Key-value pairs for fast lookups and structured data

**Use in PySpark**: Used for mapping column names, configurations, or creating row-like structures.

Example: config = {"spark.master": "local", "appName": "PySparkApp"} print(config["appName"]) # Output: PySparkApp

1. **Sets**: Unordered collections of unique items. Useful for deduplication and membership tests.

**Summary**: Sets are optimized for checking unique values and removing duplicates from large datasets.

Example: data = {1, 2, 3, 3, 4} # Set removes duplicate '3' print(data) # Output: {1, 2, 3, 4}

1. **Tuples**: Immutable, ordered collections of items. Useful for fixed data structures. **Summary**: Tuples are faster than lists and ideal for storing unchangeable data like schema definitions.

Example: schema = ("id", "name", "age") # Tuple representing column names print(schema[1]) # Output: "name"