**Spark Key Point**

1. **SparkSession**
2. **SparkContext**
3. **SQLContext**
4. **RDD**
5. **Dataset**
6. **DataFrame**
7. **Example of Dataset (Java)**
8. **Example of DataFrame (Java, Python)**
9. **PySpark SQL Functions**
10. **PySpark SQL Window**
11. **RDD to DataFrame**
12. **RDD Functions**
13. **DataFrame Functions**
14. **When Functions in DataFrame**

**SparkSession**

A unified entry point to Spark functionalities introduced in Spark 2.0. It combines the functionality of SparkContext and SQLContext, making them accessible in one object.

* Provides a single interface for accessing Spark functionalities (e.g., RDDs, DataFrames, SQL).
* Used for creating and working with structured and unstructured data.
* Accesses SparkContext and SQLContext internally.

**SparkContext**

The core entry point for Spark applications; it connects the application to the Spark cluster and coordinates resource allocation.

* **Cluster Connection**: It connects to a Spark cluster, either locally or on a distributed cluster, to submit Spark jobs.
* **Resource Management**: It is responsible for acquiring resources (such as executors and cores) on the cluster for running the application.
* **Job Scheduling**: It schedules jobs and tasks to be executed on different nodes in the cluster, ensuring efficient parallel computation.
* **RDD Creation**: Enables the creation of RDDs from data sources.
* **Configuration**: Holds configuration settings for the application.
* **Access to Spark Services**: Provides access to SQLContext, StreamingContext, etc.
* **Job Monitoring**: Tracks the execution status and progress of jobs and tasks.

**SQLContext**

An interface for working with structured data, **built on top of SparkContext**. (e.g., **DataFrames** and **SQL queries**) using Spark's **SQL** and **DataFrame** APIs. It allows users to execute SQL queries against data, as well as load and save data in a variety of formats (e.g., Parquet, JSON, JDBC).

* **SQL Queries**: Enables executing SQL queries over structured data (e.g., DataFrames, Hive tables).
* **DataFrame Creation**: Allows creating DataFrames from various data sources, such as CSV, Parquet, JDBC, etc.
* **Access to Spark SQL**: Provides integration with Spark SQL to run SQL queries.
* **Hive Support**: SQLContext can connect to Apache Hive, enabling querying of Hive tables.
* **Data Manipulation**: Facilitates data manipulation using SQL-like queries on DataFrames.

**Different Between SparkSession, SparkContext, and SQLContext**

| **Aspect** | **SparkSession** | **SparkContext** | **SQLContext** |
| --- | --- | --- | --- |
| **Introduced In** | Spark 2.0 | Spark 1.x | Spark 1.x |
| **Purpose** | Unified entry point for all operations. | Core engine for Spark functionality. | Interface for structured data and SQL. |
| **Replaces** | Combines SQLContext and HiveContext. | - | Replaced by SparkSession. |
| **Primary Use Case** | High-level operations with DataFrames, SQL, and datasets. | Low-level operations with RDDs. | Querying structured data. |
| **Access** | Unified, includes SparkContext and SQLContext. | Accessed directly or via SparkSession. | Accessed via SparkSession or explicitly created. |
| **Recommendation** | Use SparkSession for modern applications. | Use indirectly through SparkSession. | Deprecated; use SparkSession. |

**RDD (Resilient Distributed Dataset)**

RDD is the fundamental data abstraction in Apache Spark. It is an immutable, distributed collection of objects that can be processed in parallel across a cluster.

* **Resilient**: Fault-tolerant, able to recover from node failures using lineage (a record of transformations applied to produce the RDD).
* **Distributed**: Data is partitioned and distributed across the nodes in the cluster.
* **Immutable**: Once created, an RDD cannot be modified. You can only transform it to create a new RDD.
* **Lazy Evaluation**: Transformations are not executed until an action is triggered. This optimizes the execution plan.
* **Partitioned**: Data in RDDs is divided into partitions for parallel processing.

**How RDDs Work**

RDDs allow you to perform two types of operations:

* **Transformations**: Operations that create a new RDD from an existing one (e.g., map, filter, flatMap).
* **Actions**: Operations that trigger computation and return results to the driver (e.g., collect, reduce, count).
* **DataFrame and Dataset APIs, which are built on top of RDDs.**

**Dataset**

* **Java -** Dataset is a **strongly-typed** distributed collection of data, means work with custom objects (POJOs) and define schemas, which makes it possible to perform compile-time type safety checks.
* **Python –** Dataset is **not available** in PySpark. Spark’s Python API works with **DataFrames** only. Python does not support the **Dataset** abstraction that is available in Java and Scala.

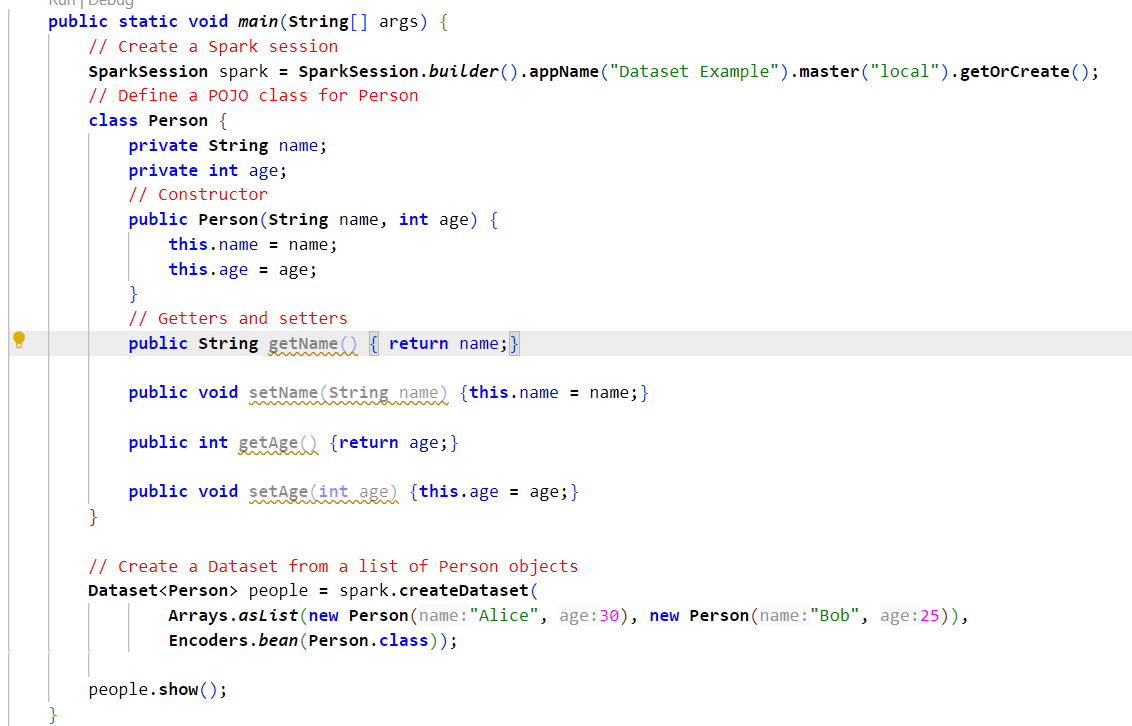
**DataFrame**

* **Java -** DataFrame is a distributed collection of data organized into named columns, represented as rows of generic Row objects. It is **weakly-typed** and suitable for SQL-like operations and tabular data processing.
* **Python -** DataFrame is a distributed collection of data in a tabular format (rows and columns). It is **weakly-typed** and is the primary abstraction for structured data processing in PySpark.

**Compare of Dataset and DataFrame**

| **Feature** | **Java (Dataset)** | **Java (DataFrame)** | **Python (DataFrame)** |
| --- | --- | --- | --- |
| **Availability** | Available (strongly-typed) | Available (weakly-typed) | Available (only weakly-typed DataFrame) |
| **Type Safety** | Strongly typed (custom POJOs) | Weakly typed (rows of generic Row) | Weakly typed (rows of generic Row) |
| **Data Representation** | Strongly typed objects (Person class) | Rows in a tabular format (no compile-time checks) | Rows in a tabular format (no compile-time checks) |
| **API** | Dataset API (POJOs and custom types) | DataFrame API (SQL-like operations) | DataFrame API (SQL-like operations) |
| **Best Use Case** | When you need compile-time type safety | When you need SQL-like operations | General data processing (only DataFrame) |
| **Interoperability** | Can convert to/from DataFrame | Can convert to/from Dataset (via Row type) | N/A (only DataFrame) |

**Example of Dataset (Java)**

****

**Output**

A white background with black text

Description automatically generated

**Example of DataFrame (Java)  
  
A screenshot of a computer program

Description automatically generated**

**Output**

A white background with text

Description automatically generated

**Example of DataFrame (Python)  
  
**

**PySpark SQL Functions**

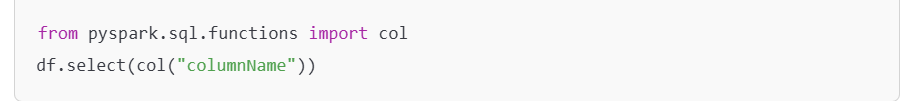
**pyspark.sql.functions** is a module in PySpark that provides a collection of built-in functions for manipulating and transforming data in DataFrames. These functions are designed for working with columns and provide various capabilities, such as data manipulation, type conversion, aggregation, and more.

1. **Column Creation and Manipulation**
   1. **lit(value):** Creates a column with a literal value, that means constant value

**A white background with black text

Description automatically generated**

* 1. **col(columnName):** Refers to a column by name.

****

* 1. **when(condition, value):** Similar to SQL CASE, used for conditional logic.

**A close-up of a number

Description automatically generated**

* 1. **row\_number :** used for ordering the value of column



* 1. **monotonically\_increasing\_id :** used for creating sequence number in ordering the value of column

**A screenshot of a computer program

Description automatically generated**

1. **Aggregation**
   1. **sum(column):** Sum of a column

**A computer screen shot of a computer screen

Description automatically generated**

* 1. **avg(column) / mean(column):** Average of a column

****

* 1. **max(column) / min(column):** Maximum or minimum value in a column

**A close-up of a computer code

Description automatically generated**

1. **String Manipulation**
   1. **concat(\*cols):** Concatenates multiple columns into one

****

* 1. **lower(column) / upper(column):** Converts strings to lowercase or uppercase

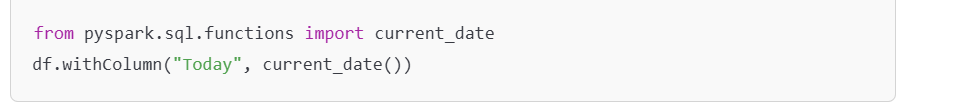
****

* 1. **length(column):** Returns the length of a string

**A close-up of a computer code

Description automatically generated**

1. **Date and Time Functions**
   1. **current\_date() / current\_timestamp():** Returns the current date or timestamp

****

* 1. **to\_date(column):** Converts a string to a date

****

* 1. **datediff(endDate, startDate):** Difference between two dates

**A close-up of a computer code

Description automatically generated**

1. **Null Handling**
   1. **isnull(column) / isnotnull(column):** Checks if a column has null or not-null values

**A close-up of a number

Description automatically generated**

* 1. **coalesce(\*cols):** Returns the first non-null value

**A close-up of a computer code

Description automatically generated**

1. **Type Conversion**
   1. **cast(dataType):** Converts a column to a specified data type

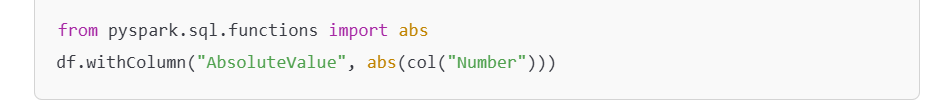
****

* 1. **to\_timestamp(column):** Converts a string to a timestamp

**A close-up of a text

Description automatically generated**

1. **Mathematical Functions**
   1. **abs(column):** Absolute value of a column

****

* 1. **round(column, scale):** Rounds the values in a column to a specified number of decimal places

****

* 1. **sqrt(column):** Square root of a column

****

1. **JSON and Complex Data Types**
   1. **explode(column):** Creates a new row for each element in an array or map

****

* 1. **to\_json(column) / from\_json(column, schema):** Converts a column to or from JSON format

****

1. **User-Defined Functions (UDFs)**

**PySpark SQL Window**

A PySpark Window is a function that allows you to perform operations over a specified range of rows in your dataset, defined by a "window specification." This is useful for tasks like calculating running totals, moving averages, ranking rows, and more.

* **Partition**: Defines how the data is grouped. Each group is processed independently.
* **Order**: Specifies the order of rows within each partition.
* **Frame**: Defines the range of rows relative to the current row being processed.

Example of window function  
A screenshot of a computer program

Description automatically generated

A white background with black dots

Description automatically generated

**RDD to DataFrame**

* **Object to DataFrame**

**A screenshot of a computer program

Description automatically generated**

* **List to DataFrame**

**A screenshot of a computer program

Description automatically generated**

**RDD Functions**

**RDD Transformation Functions**

* **map(func):** Applies the func to each element of the RDD and returns a new RDD.

A white background with black and yellow text

Description automatically generated

* **flatMap(func):** Similar to map(), but it flattens the result.

**A screenshot of a computer code

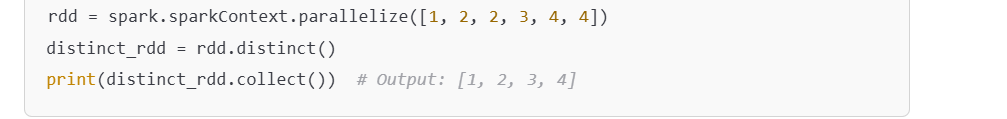
Description automatically generated**

* **filter(func):** Returns an RDD containing only elements that satisfy the func.

**A number and numbers on a white background

Description automatically generated**

* **distinct():** Removes duplicates from the RDD.

****

* **union(otherRDD):** Combines two RDDs.

A white background with black and yellow text

Description automatically generated

* **intersection(otherRDD):** Returns elements that are common between two RDDs.

**A white background with black and yellow text

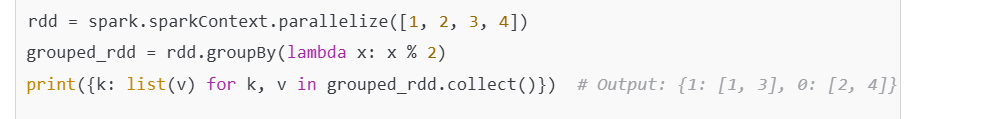
Description automatically generated**

* **cartesian(otherRDD):** Computes the Cartesian product of two RDDs

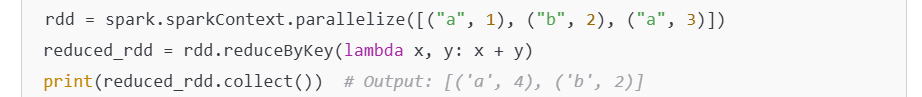
**A white background with black and green text

Description automatically generated**

* **groupBy(func):** Groups elements based on the result of the function.

****

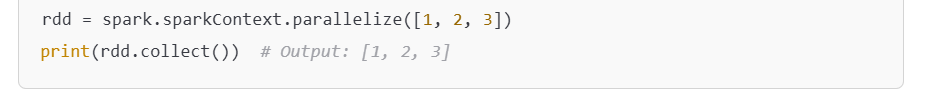
* **reduceByKey(func) (for Pair RDDs):** Combines values with the same key**.**

****

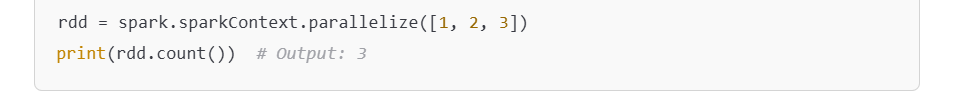
* **sortBy(func):** Sorts the RDD by the result of the function.
* Sum(), Substract() etc.

**RDD Actions Functions**

* **collect():** Retrieves all elements of the RDD.

****

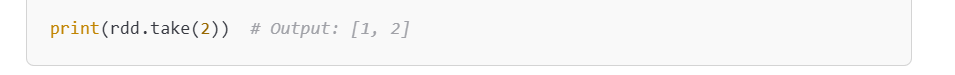
* **count():** Returns the number of elements in the RDD.

****

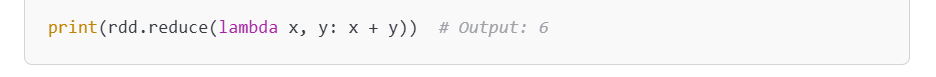
* **first():** Returns the first element.



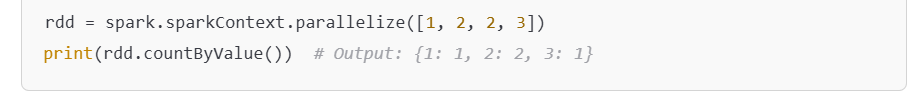
* **take(n):** Returns the first n elements.

****

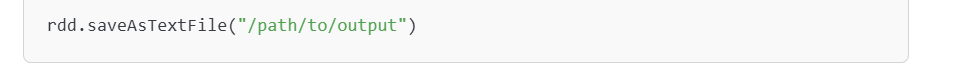
* **reduce(func):** Aggregates elements using func.

****

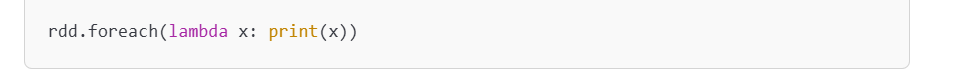
* **countByValue():** Counts the occurrence of each unique value.



* **saveAsTextFile(path):** Saves the RDD as a text file.

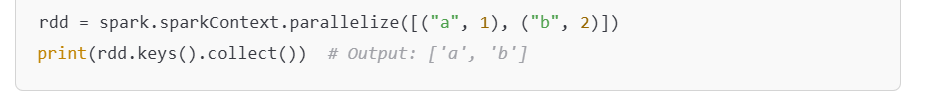
****

* **foreach(func):** Applies func to each element without returning a result.

****

**RDD Key-Value Pair-Specific Functions**

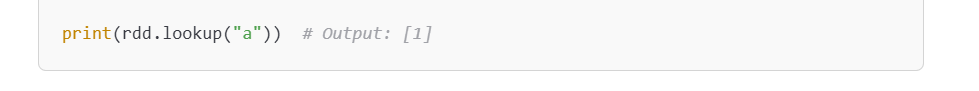
* **keys(): Extracts keys from the RDD.**

****

* **values(): Extracts values from the RDD.**

****

* **lookup(key): Returns all values associated with a key.**

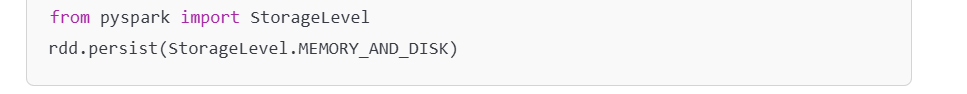
****

**RDD Persistence/Cache Functions**

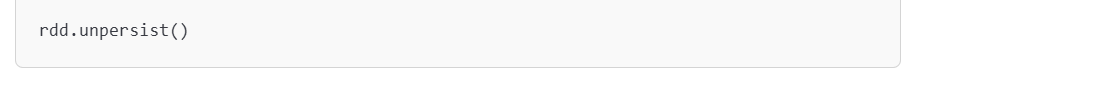
* **cache():** Persists the RDD in memory.



* **persist(storageLevel):** Allows finer control over persistence.

****

* **unpersist():** Removes the RDD from memory/disk.



**DataFrame Functions**

1. **Creating a DataFrame**

**A screenshot of a computer code

Description automatically generated**

1. **Basic Operations**
   * **Display Data**

**A white background with black text

Description automatically generated**

* + **Schema and Structure**

**A close-up of a white background

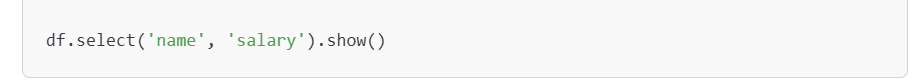
Description automatically generated**

* + **Summary Statistics**

**A close-up of a white background

Description automatically generated**

1. **Selecting Data**
   * **Select Specific Columns**

****

* + **Adding New Columns**

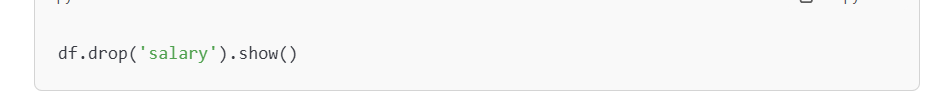
**A screenshot of a computer code

Description automatically generated**

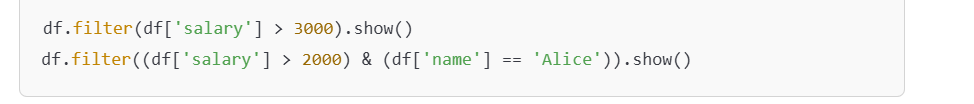
* + **Renaming Columns**

****

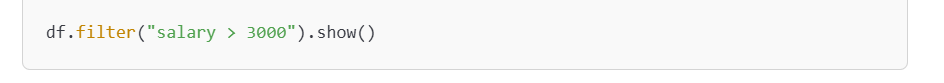
* + **Dropping Columns**

****

1. **Filtering Rows**
   * **Using Conditions**

****

* + **Using SQL-like Syntax**

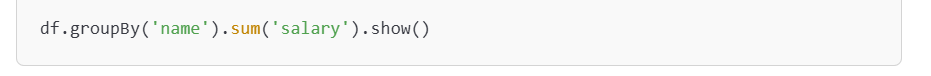
****

1. **Aggregation and Grouping**
   * **Basic Aggregation**

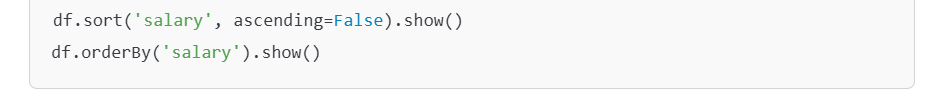
**A close up of text

Description automatically generated**

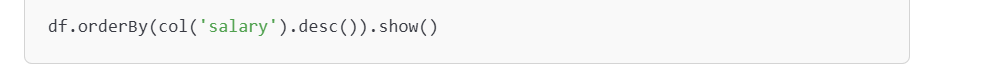
* + **Group By**

****

1. **Sorting and Ordering**
   * **Sort by Column**

****

* + **Sort with Expressions**

****

1. **Joins**
   * **Joining Two DataFrames**

**A screenshot of a computer code

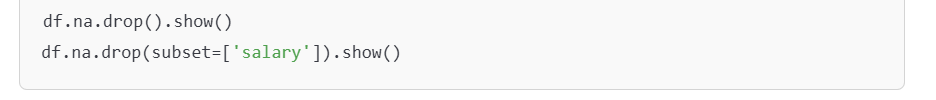
Description automatically generated**

1. **SQL Queries**
   * **Register as a Temporary Table**

**A close-up of a white background

Description automatically generated**

1. **Handling Null Values**
   * **Dropping Nulls**

****

* + **Filling Nulls**

**A close-up of a computer code

Description automatically generated**

1. **Working with UDFs (User-Defined Functions)**
   * **Define and Apply UDF**

**A screenshot of a computer code

Description automatically generated**

1. **Saving and Loading Data**
   * **Save to Disk**

****

* + **Read from Disk**

**A close up of a computer screen

Description automatically generated**

**When functions in DataFrame (it is used for to replace value by apply condition)**

Syntax - when(*condition: pyspark.sql.column.Column*, *value: Any*)

[**pyspark.sql.Column.otherwise()**](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.Column.otherwise.html#pyspark.sql.Column.otherwise) is not invoked, None is returned for unmatched conditions.

Example –

