### **1. What is PySpark?**

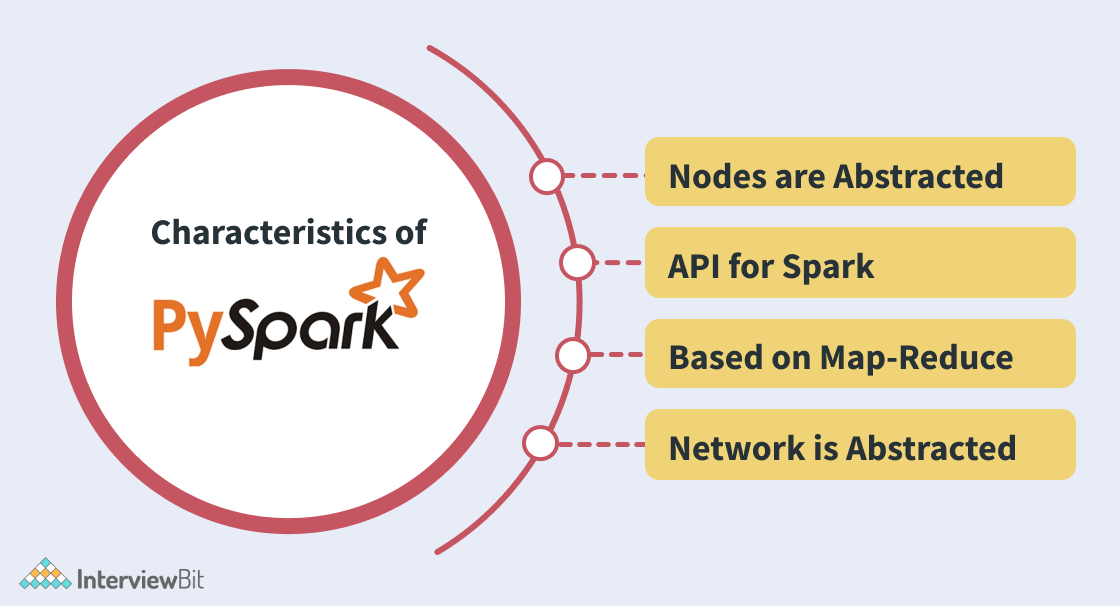
PySpark is an Apache Spark interface in Python. It is used for collaborating with Spark using APIs written in Python. It also supports Spark’s features like Spark DataFrame, Spark SQL, Spark Streaming, Spark MLlib and Spark Core. It provides an interactive PySpark shell to analyze structured and semi-structured data in a distributed environment. PySpark supports reading data from multiple sources and different formats. It also facilitates the use of RDDs (Resilient Distributed Datasets). PySpark features are implemented in the py4j library in python.

PySpark can be installed using PyPi by using the command:

pip install pyspark

### **2. What are the characteristics of PySpark?**

There are 4 characteristics of PySpark:



* ****Abstracted Nodes:**** This means that the individual worker nodes can not be addressed.
* ****Spark API:**** PySpark provides APIs for utilizing Spark features.
* ****Map-Reduce Model:**** PySpark is based on Hadoop’s Map-Reduce model this means that the programmer provides the map and the reduce functions.
* ****Abstracted Network:**** Networks are abstracted in PySpark which means that the only possible communication is implicit communication.

### **3. What are the advantages and disadvantages of PySpark?**

****Advantages of PySpark:****

* Simple to use: Parallelized code can be written in a simpler manner.
* Error Handling: PySpark framework easily handles errors.
* Inbuilt Algorithms: PySpark provides many of the useful algorithms in Machine Learning or Graphs.
* Library Support: Compared to Scala, Python has a huge library collection for working in the field of data science and data visualization.
* Easy to Learn: PySpark is an easy to learn language.

****Disadvantages of PySpark:****

* Sometimes, it becomes difficult to express problems using the MapReduce model.
* Since Spark was originally developed in Scala, while using PySpark in Python programs they are relatively less efficient and approximately 10x times slower than the Scala programs. This would impact the performance of heavy data processing applications.
* The Spark Streaming API in PySpark is not mature when compared to Scala. It still requires improvements.
* PySpark cannot be used for modifying the internal function of the Spark due to the abstractions provided. In such cases, Scala is preferred.

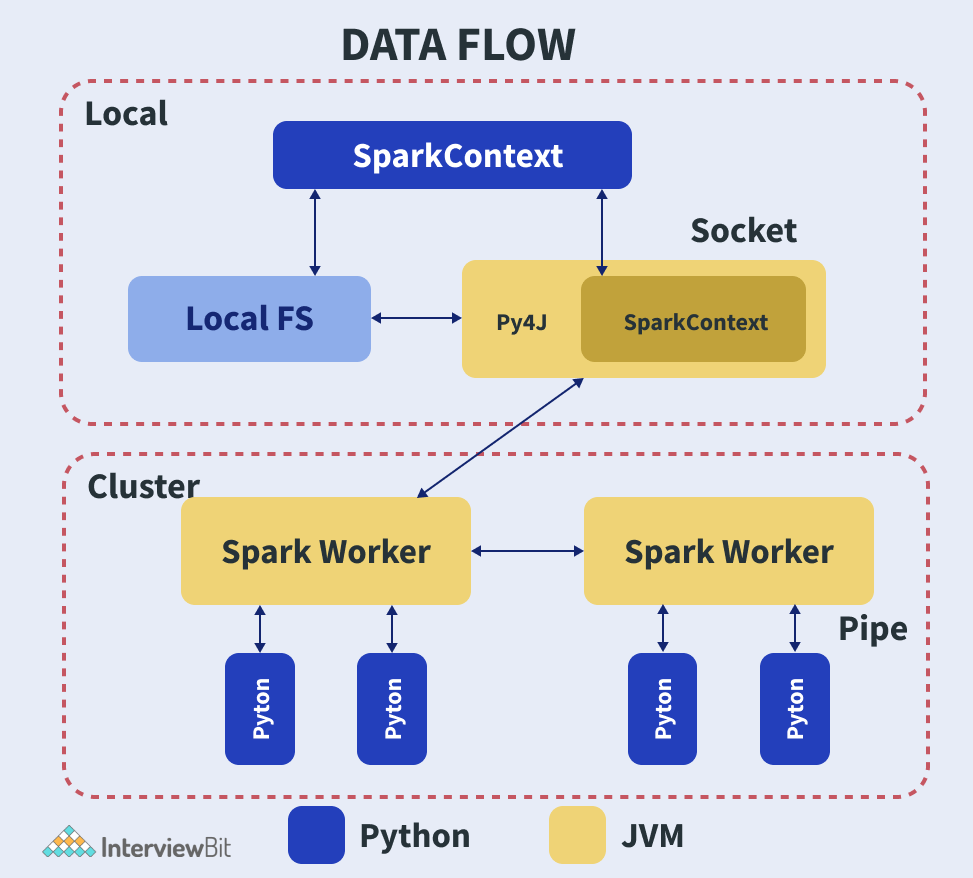
**You can download a PDF version of Pyspark Interview Questions.**

**[IMG_257Download PDF](javascript:void(0))**

### **4. What is PySpark SparkContext?**

PySpark SparkContext is an initial entry point of the spark functionality. It also represents Spark Cluster Connection and can be used for creating the Spark RDDs (Resilient Distributed Datasets) and broadcasting the variables on the cluster.

The following diagram represents the architectural diagram of PySpark’s SparkContext:



When we want to run the Spark application, a driver program that has the main function will be started. From this point, the SparkContext that we defined gets initiated. Later on, the driver program performs operations inside the executors of the worker nodes. Additionally, JVM will be launched using Py4J which in turn creates JavaSparkContext. Since PySpark has default SparkContext available as “sc”, there will not be a creation of a new SparkContext.

### **5. Why do we use PySpark SparkFiles?**

PySpark’s SparkFiles are used for loading the files onto the Spark application. This functionality is present under SparkContext and can be called using the sc.addFile() method for loading files on Spark. SparkFiles can also be used for getting the path using the SparkFiles.get() method. It can also be used to resolve paths to files added using the sc.addFile() method.

### **6. What are PySpark serializers?**

The serialization process is used to conduct performance tuning on Spark. The data sent or received over the network to the disk or memory should be persisted. PySpark supports serializers for this purpose. It supports two types of serializers, they are:

* ****PickleSerializer:**** This serializes objects using Python’s PickleSerializer (class pyspark.PickleSerializer). This supports almost every Python object.
* ****MarshalSerializer:**** This performs serialization of objects. We can use it by using class pyspark.MarshalSerializer. This serializer is faster than the PickleSerializer but it supports only limited types.

Consider an example of serialization which makes use of MarshalSerializer:

# --serializing.py----**from** pyspark.context **import** SparkContext**from** pyspark.serializers **import** MarshalSerializer

sc = SparkContext("local", "Marshal Serialization", serializer = MarshalSerializer()) #Initialize spark context and serializerprint(sc.parallelize(list(range(1000))).map(**lambda** x: 3 \* x).take(5))

sc.stop()

When we run the file using the command:

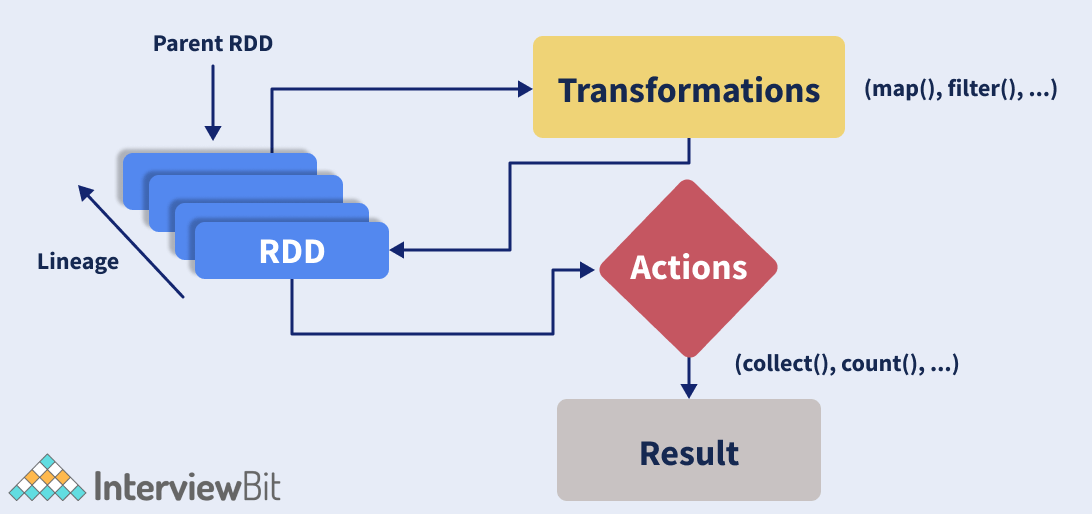
$SPARK\_HOME/bin/spark-submit serializing.py

The output of the code would be the list of size 5 of numbers multiplied by 3:

[0, 3, 6, 9, 12]

### **7. What are RDDs in PySpark?**

RDDs expand to Resilient Distributed Datasets. These are the elements that are used for running and operating on multiple nodes to perform parallel processing on a cluster. Since RDDs are suited for parallel processing, they are immutable elements. This means that once we create RDD, we cannot modify it. RDDs are also fault-tolerant which means that whenever failure happens, they can be recovered automatically. Multiple operations can be performed on RDDs to perform a certain task. The operations can be of 2 types:



* ****Transformation:**** These operations when applied on RDDs result in the creation of a new RDD. Some of the examples of transformation operations are filter, groupBy, map.  
  Let us take an example to demonstrate transformation operation by considering filter() operation:

**from** pyspark **import** SparkContext

sc = SparkContext("local", "Transdormation Demo")

words\_list = sc.parallelize (

["pyspark",

"interview",

"questions",

"at",

"interviewbit"]

)

filtered\_words = words\_list.filter(**lambda** x: 'interview' **in** x)

filtered = filtered\_words.collect()print(filtered)

The above code filters all the elements in the list that has ‘interview’ in the element. The output of the above code would be:

[

"interview",

"interviewbit"

]

* ****Action:**** These operations instruct Spark to perform some computations on the RDD and return the result to the driver. It sends data from the Executer to the driver. count(), collect(), take() are some of the examples.  
  Let us consider an example to demonstrate action operation by making use of the count() function.

**from** pyspark **import** SparkContext

sc = SparkContext("local", "Action Demo")

words = sc.parallelize (

["pyspark",

"interview",

"questions",

"at",

"interviewbit"]

)

counts = words.count()print("Count of elements in RDD -> ", counts)

In this class, we count the number of elements in the spark RDDs. The output of this code is

Count of elements **in** RDD -> 5

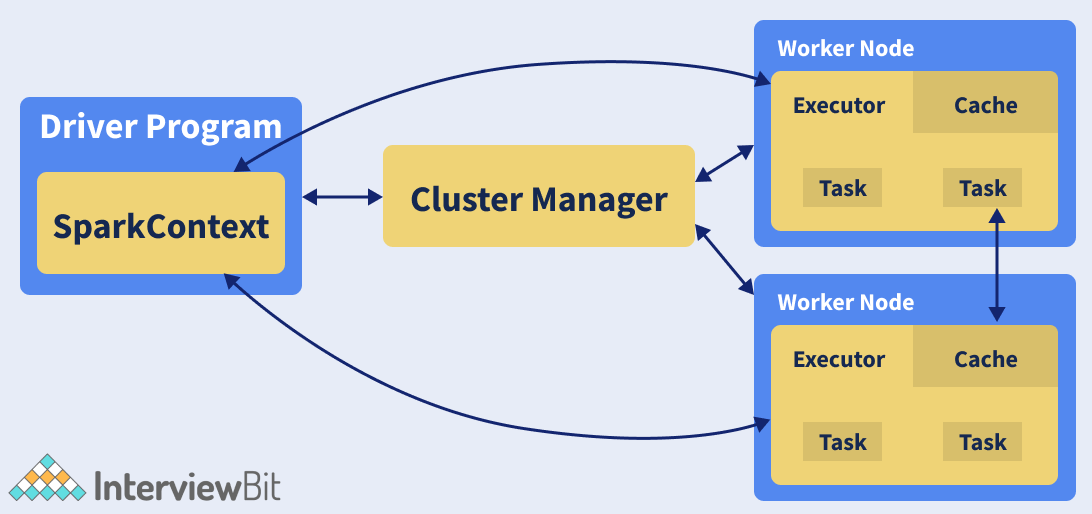
### **8. Does PySpark provide a machine learning API?**

Similar to Spark, PySpark provides a machine learning API which is known as MLlib that supports various ML algorithms like:

* mllib.classification − This supports different methods for binary or multiclass classification and regression analysis like Random Forest, Decision Tree, Naive Bayes etc.
* mllib.clustering − This is used for solving clustering problems that aim in grouping entities subsets with one another depending on similarity.
* mllib.fpm − FPM stands for Frequent Pattern Matching. This library is used to mine frequent items, subsequences or other structures that are used for analyzing large datasets.
* mllib.linalg − This is used for solving problems on linear algebra.
* mllib.recommendation − This is used for collaborative filtering and in recommender systems.
* spark.mllib − This is used for supporting model-based collaborative filtering where small latent factors are identified using the Alternating Least Squares (ALS) algorithm which is used for predicting missing entries.
* mllib.regression − This is used for solving problems using regression algorithms that find relationships and variable dependencies.

### **9. What are the different cluster manager types supported by PySpark?**

A cluster manager is a cluster mode platform that helps to run Spark by providing all resources to worker nodes based on the requirements.



The above figure shows the position of cluster manager in the Spark ecosystem. Consider a master node and multiple worker nodes present in the cluster. The master nodes provide the worker nodes with the resources like memory, processor allocation etc depending on the nodes requirements with the help of the cluster manager.

PySpark supports the following cluster manager types:

* ****Standalone**** – This is a simple cluster manager that is included with Spark.
* ****Apache Mesos**** – This manager can run Hadoop MapReduce and PySpark apps.
* ****Hadoop YARN**** – This manager is used in Hadoop2.
* ****Kubernetes**** – This is an open-source cluster manager that helps in automated deployment, scaling and automatic management of containerized apps.
* ****local**** – This is simply a mode for running Spark applications on laptops/desktops.

### **10. What are the advantages of PySpark RDD?**

PySpark RDDs have the following advantages:

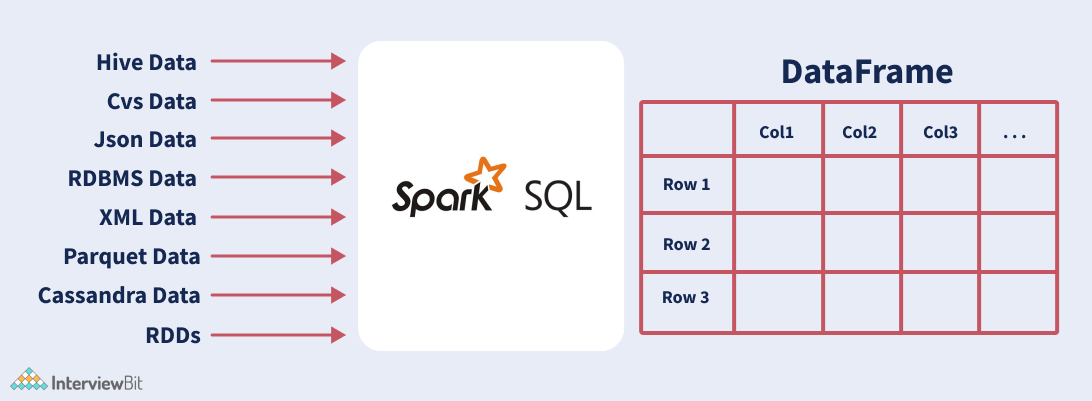
* ****In-Memory Processing:****PySpark’s RDD helps in loading data from the disk to the memory. The RDDs can even be persisted in the memory for reusing the computations.
* ****Immutability:****The RDDs are immutable which means that once created, they cannot be modified. While applying any transformation operations on the RDDs, a new RDD would be created.
* ****Fault Tolerance:****The RDDs are fault-tolerant. This means that whenever an operation fails, the data gets automatically reloaded from other available partitions. This results in seamless execution of the PySpark applications.
* ****Lazy Evolution:****The PySpark transformation operations are not performed as soon as they are encountered. The operations would be stored in the DAG and are evaluated once it finds the first RDD action.
* ****Partitioning:****Whenever RDD is created from any data, the elements in the RDD are partitioned to the cores available by default.

### **11. Is PySpark faster than pandas?**

PySpark supports parallel execution of statements in a distributed environment, i.e on different cores and different machines which are not present in Pandas. This is why PySpark is faster than pandas.

### **12. What do you understand about PySpark DataFrames?**

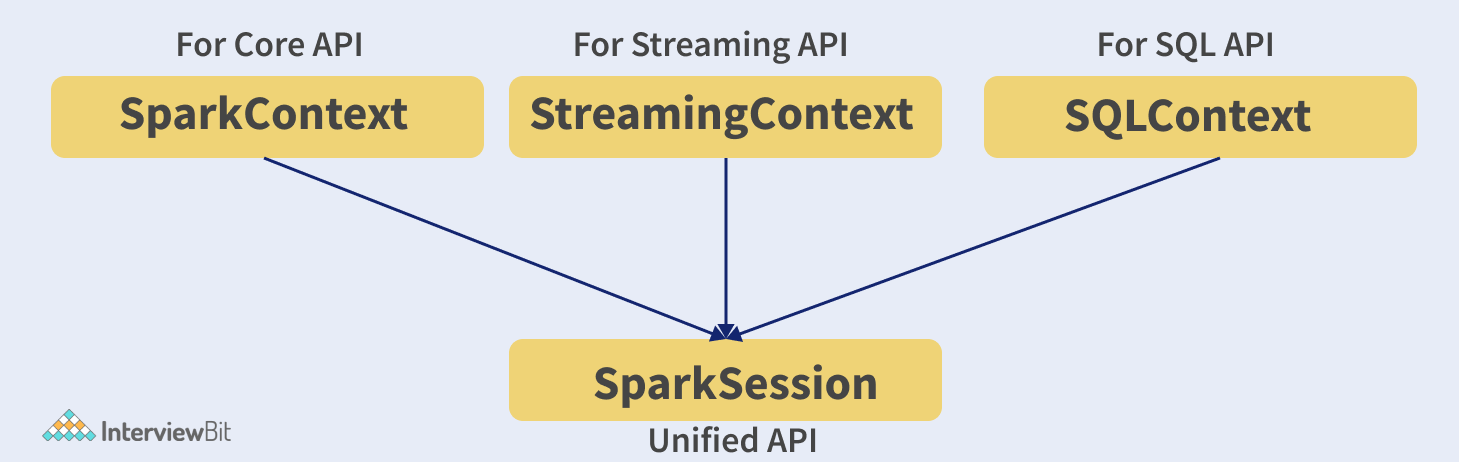
PySpark DataFrame is a distributed collection of well-organized data that is equivalent to tables of the relational databases and are placed into named columns. PySpark DataFrame has better optimisation when compared to R or python. These can be created from different sources like Hive Tables, Structured Data Files, existing RDDs, external databases etc as shown in the image below:



The data in the PySpark DataFrame is distributed across different machines in the cluster and the operations performed on this would be run parallelly on all the machines. These can handle a large collection of structured or semi-structured data of a range of petabytes.

### **13. What is SparkSession in Pyspark?**

SparkSession is the entry point to PySpark and is the replacement of SparkContext since PySpark version 2.0. This acts as a starting point to access all of the PySpark functionalities related to RDDs, DataFrame, Datasets etc. It is also a Unified API that is used in replacing the SQLContext, StreamingContext, HiveContext and all other contexts.



The SparkSession internally creates SparkContext and SparkConfig based on the details provided in SparkSession. SparkSession can be created by making use of builder patterns.

### **14. What are the types of PySpark’s shared variables and why are they useful?**

Whenever PySpark performs the transformation operation using filter(), map() or reduce(), they are run on a remote node that uses the variables shipped with tasks. These variables are not reusable and cannot be shared across different tasks because they are not returned to the Driver. To solve the issue of reusability and sharing, we have shared variables in PySpark. There are two types of shared variables, they are:

****Broadcast variables:**** These are also known as read-only shared variables and are used in cases of data lookup requirements. These variables are cached and are made available on all the cluster nodes so that the tasks can make use of them. The variables are not sent with every task. They are rather distributed to the nodes using efficient algorithms for reducing the cost of communication. When we run an RDD job operation that makes use of Broadcast variables, the following things are done by PySpark:

* The job is broken into different stages having distributed shuffling. The actions are executed in those stages.
* The stages are then broken into tasks.
* The broadcast variables are broadcasted to the tasks if the tasks need to use it.

Broadcast variables are created in PySpark by making use of the broadcast(variable) method from the SparkContext class. The syntax for this goes as follows:

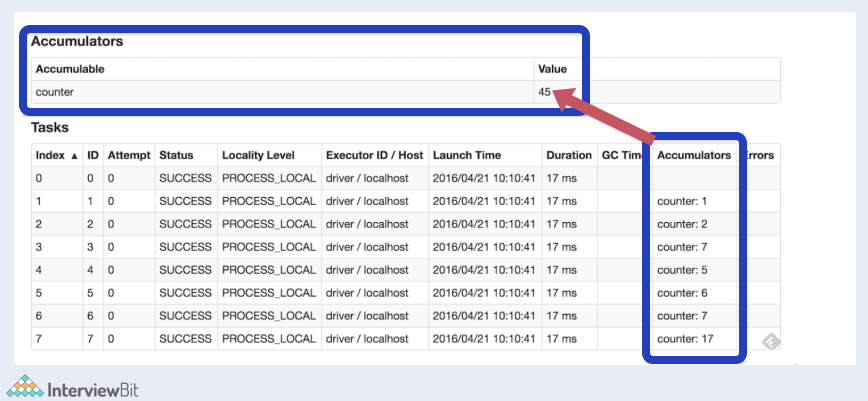
broadcastVar = sc.broadcast([10, 11, 22, 31])

broadcastVar.value # access broadcast variable

An important point of using broadcast variables is that the variables are not sent to the tasks when the broadcast function is called. They will be sent when the variables are first required by the executors.

****Accumulator variables:**** These variables are called updatable shared variables. They are added through associative and commutative operations and are used for performing counter or sum operations. PySpark supports the creation of numeric type accumulators by default. It also has the ability to add custom accumulator types. The custom types can be of two types:

* ****Named Accumulators****: These accumulators are visible under the “Accumulator” tab in the PySpark web UI as shown in the image below:



Here, we will see the Accumulable section that has the sum of the Accumulator values of the variables modified by the tasks listed in the Accumulator column present in the Tasks table.

* ****Unnamed Accumulators:**** These accumulators are not shown on the PySpark Web UI page. It is always recommended to make use of named accumulators.

Accumulator variables can be created by using SparkContext.longAccumulator(variable) as shown in the example below:

ac = sc.longAccumulator("sumaccumulator")

sc.parallelize([2, 23, 1]).foreach(lambda x: ac.add(x))

Depending on the type of accumulator variable data - double, long and collection, PySpark provide DoubleAccumulator, LongAccumulator and CollectionAccumulator respectively.

### **15. What is PySpark UDF?**

UDF stands for User Defined Functions. In PySpark, UDF can be created by creating a python function and wrapping it with PySpark SQL’s udf() method and using it on the DataFrame or SQL. These are generally created when we do not have the functionalities supported in PySpark’s library and we have to use our own logic on the data. UDFs can be reused on any number of SQL expressions or DataFrames.

### **16. What are the industrial benefits of PySpark?**

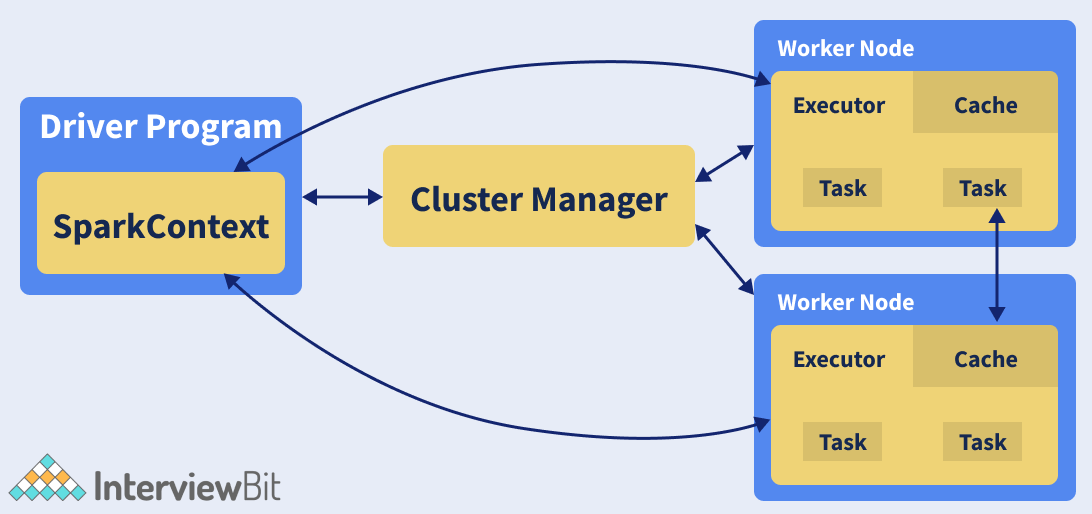
These days, almost every industry makes use of big data to evaluate where they stand and grow. When you hear the term big data, Apache Spark comes to mind. Following are the industry benefits of using PySpark that supports Spark:

* ****Media streaming:**** Spark can be used to achieve real-time streaming to provide personalized recommendations to subscribers. Netflix is one such example that uses Apache Spark. It processes around 450 billion events every day to flow to its server-side apps.
* ****Finance:**** Banks use Spark for accessing and analyzing the social media profiles and in turn get insights on what strategies would help them to make the right decisions regarding customer segmentation, credit risk assessments, early fraud detection etc.
* ****Healthcare:**** Providers use Spark for analyzing the past records of the patients to identify what health issues the patients might face posting their discharge. Spark is also used to perform genome sequencing for reducing the time required for processing genome data.
* ****Travel Industry:**** Companies like TripAdvisor uses Spark to help users plan the perfect trip and provide personalized recommendations to the travel enthusiasts by comparing data and review from hundreds of websites regarding the place, hotels, etc.
* ****Retail and e-commerce:**** This is one important industry domain that requires big data analysis for targeted advertising. Companies like Alibaba run Spark jobs for analyzing petabytes of data for enhancing customer experience, providing targetted offers, sales and optimizing the overall performance.

## **Pyspark Interview Questions for Experienced**

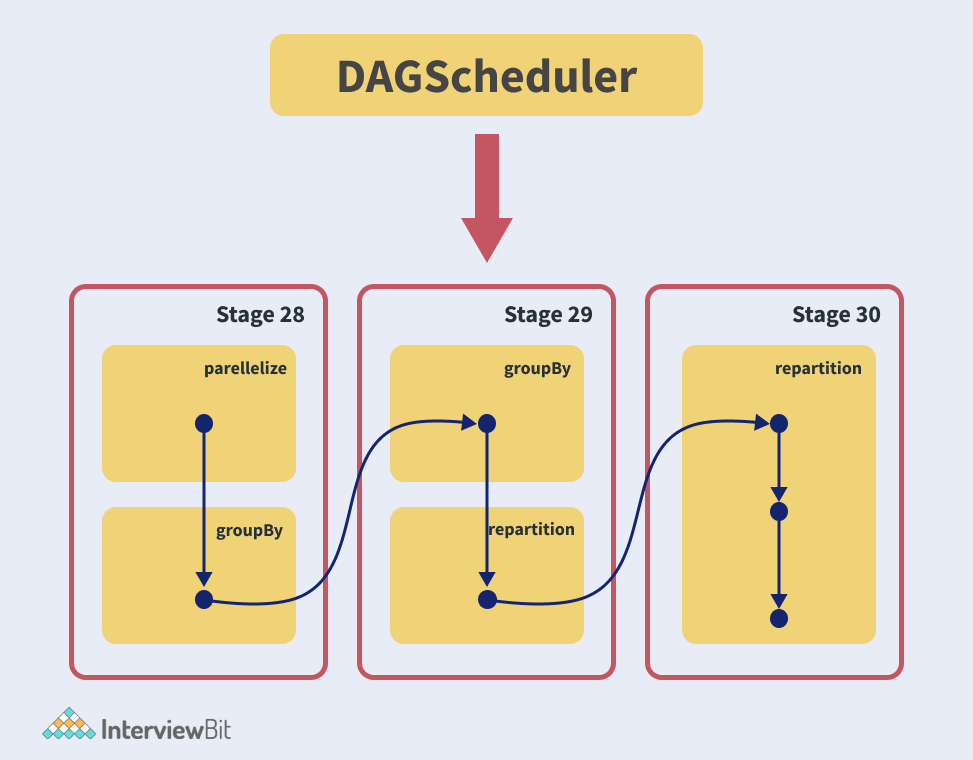
### **17. What is PySpark Architecture?**

PySpark similar to Apache Spark works in master-slave architecture pattern. Here, the master node is called the Driver and the slave nodes are called the workers. When a Spark application is run, the Spark Driver creates SparkContext which acts as an entry point to the spark application. All the operations are executed on the worker nodes. The resources required for executing the operations on the worker nodes are managed by the Cluster Managers. The following diagram illustrates the architecture described:



### **18. What PySpark DAGScheduler?**

DAG stands for Direct Acyclic Graph. DAGScheduler constitutes the scheduling layer of Spark which implements scheduling of tasks in a stage-oriented manner using jobs and stages. The logical execution plan (Dependencies lineage of transformation actions upon RDDs) is transformed into a physical execution plan consisting of stages. It computes a DAG of stages needed for each job and keeps track of what stages are RDDs are materialized and finds a minimal schedule for running the jobs. These stages are then submitted to TaskScheduler for running the stages. This is represented in the image flow below:



DAGScheduler performs the following three things in Spark:

* Compute DAG execution for the job.
* Determine preferred locations for running each task
* Failure Handling due to output files lost during shuffling.

PySpark’s DAGScheduler follows event-queue architecture. Here a thread posts events of type DAGSchedulerEvent such as new stage or job. The DAGScheduler then reads the stages and sequentially executes them in topological order.

### **19. What is the common workflow of a spark program?**

The most common workflow followed by the spark program is:

* The first step is to create input RDDs depending on the external data. Data can be obtained from different data sources.
* Post RDD creation, the RDD transformation operations like filter() or map() are run for creating new RDDs depending on the business logic.
* If any intermediate RDDs are required to be reused for later purposes, we can persist those RDDs.
* Lastly, if any action operations like first(), count() etc are present then spark launches it to initiate parallel computation.

### **20. Why is PySpark SparkConf used?**

PySpark SparkConf is used for setting the configurations and parameters required to run applications on a cluster or local system. The following class can be executed to run the SparkConf:

**class** **pyspark**.**Sparkconf**(

localdefaults = True,

\_jvm = None,

\_jconf = None)

where:

* loadDefaults - is of type boolean and indicates whether we require loading values from Java System Properties. It is True by default.
* \_jvm - This belongs to the class py4j.java\_gateway.JVMView and is an internal parameter that is used for passing the handle to JVM. This need not be set by the users.
* \_jconf - This belongs to the class py4j.java\_gateway.JavaObject. This parameter is an option and can be used for passing existing SparkConf handles for using the parameters.

### **21. How will you create PySpark UDF?**

Consider an example where we want to capitalize the first letter of every word in a string. This feature is not supported in PySpark. We can however achieve this by creating a UDF capitalizeWord(str) and using it on the DataFrames. The following steps demonstrate this:

* Create Python function capitalizeWord that takes a string as input and capitalizes the first character of every word.

**def** **capitalizeWord**(str):

result=""

words = str.split(" ")

**for** word **in** words:

result= result + word[0:1].upper() + word[1:len(x)] + " "

**return** result

* Register the function as a PySpark UDF by using the udf() method of org.apache.spark.sql.functions.udf package which needs to be imported. This method returns the object of class org.apache.spark.sql.expressions.UserDefinedFunction.

""" Converting function to UDF """

capitalizeWordUDF = udf(**lambda** z: capitalizeWord(z),StringType())

* Use UDF with DataFrame: The UDF can be applied on a Python DataFrame as that acts as the built-in function of DataFrame.  
  Consider we have a DataFrame of stored in variable df as below:

+----------+-----------------+

|ID\_COLUMN |NAME\_COLUMN |

+----------+-----------------+

|1 |harry potter |

|2 |ronald weasley |

|3 |hermoine granger |

+----------+-----------------+

To capitalize every first character of the word, we can use:

df.select(col("ID\_COLUMN"), convertUDF(col("NAME\_COLUMN"))

.alias("NAME\_COLUMN") )

.show(truncate=False)

The output of the above code would be:

+----------+-----------------+

|ID\_COLUMN |NAME\_COLUMN |

+----------+-----------------+

|1 |Harry Potter |

|2 |Ronald Weasley |

|3 |Hermoine Granger |

+----------+-----------------+

UDFs have to be designed in a way that the algorithms are efficient and take less time and space complexity. If care is not taken, the performance of the DataFrame operations would be impacted.

### **22. What are the profilers in PySpark?**

Custom profilers are supported in PySpark. These are useful for building predictive models. Profilers are useful for data review to ensure that it is valid and can be used for consumption. When we require a custom profiler, it has to define some of the following methods:

* ****profile:**** This produces a system profile of some sort.
* ****stats:**** This returns collected stats of profiling.
* ****dump:**** This dumps the profiles to a specified path.
* ****add:**** This helps to add profile to existing accumulated profile. The profile class has to be selected at the time of SparkContext creation.
* ****dump(id, path):**** This dumps a specific RDD id to the path given.

### **23. How to create SparkSession?**

To create SparkSession, we use the builder pattern. The SparkSession class from the pyspark.sql library has the getOrCreate() method which creates a new SparkSession if there is none or else it returns the existing SparkSession object. The following code is an example for creating SparkSession:

**import** pyspark**from** pyspark.sql **import** SparkSession

spark = SparkSession.builder.master("local[1]")

.appName('InterviewBitSparkSession')

.getOrCreate()

Here,

* master() – This is used for setting up the mode in which the application has to run - cluster mode (use the master name) or standalone mode. For Standalone mode, we use the local[x] value to the function, where x represents partition count to be created in RDD, DataFrame and DataSet. The value of x is ideally the number of CPU cores available.
* appName() - Used for setting the application name
* getOrCreate() – For returning SparkSession object. This creates a new object if it does not exist. If an object is there, it simply returns that.

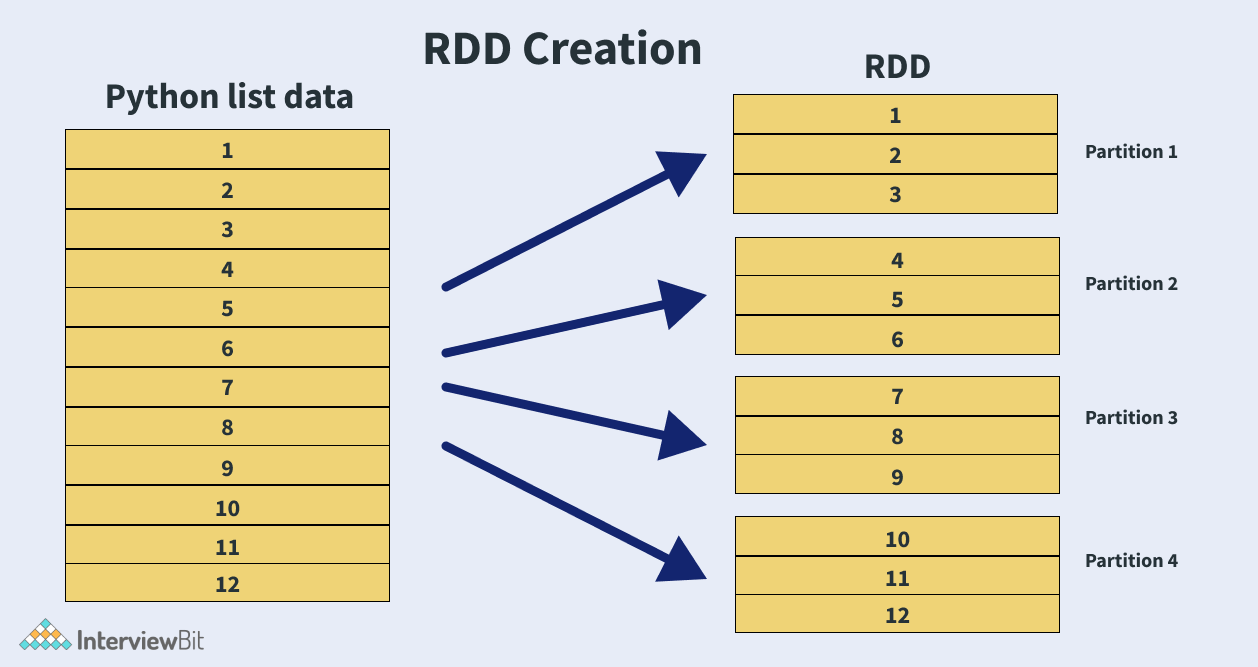
If we want to create a new SparkSession object every time, we can use the newSession method as shown below:

**import** pyspark**from** pyspark.sql **import** SparkSession

spark\_session = SparkSession.newSession

### **24. What are the different approaches for creating RDD in PySpark?**

The following image represents how we can visualize RDD creation in PySpark:



In the image, we see that the data we have is the list form and post converting to RDDs, we have it stored in different partitions.  
We have the following approaches for creating PySpark RDD:

* ****Using**** sparkContext.parallelize(): The parallelize() method of the SparkContext can be used for creating RDDs. This method loads existing collection from the driver and parallelizes it. This is a basic approach to create RDD and is used when we have data already present in the memory. This also requires the presence of all data on the Driver before creating RDD. Code to create RDD using the parallelize method for the python list shown in the image above:

list = [1,2,3,4,5,6,7,8,9,10,11,12]

rdd=spark.sparkContext.parallelize(list)

* ****Using**** sparkContext.textFile(): Using this method, we can read .txt file and convert them into RDD. Syntax:

rdd\_txt = spark.sparkContext.textFile("/path/to/textFile.txt")

* ****Using**** sparkContext.wholeTextFiles(): This function returns PairRDD (RDD containing key-value pairs) with file path being the key and the file content is the value.

#Reads entire file into a RDD as single record.

rdd\_whole\_text = spark.sparkContext.wholeTextFiles("/path/to/textFile.txt")

We can also read csv, json, parquet and various other formats and create the RDDs.

* ****Empty RDD with no partition using**** sparkContext.emptyRDD: RDD with no data is called empty RDD. We can create such RDDs having no partitions by using emptyRDD() method as shown in the code piece below:

empty\_rdd = spark.sparkContext.emptyRDD # to create empty rdd of string type

empty\_rdd\_string = spark.sparkContext.emptyRDD[String]

* ****Empty RDD with partitions using**** sparkContext.parallelize: When we do not require data but we require partition, then we create empty RDD by using the parallelize method as shown below:

#Create empty RDD with 20 partitions

empty\_partitioned\_rdd = spark.sparkContext.parallelize([],20)

### **25. How can we create DataFrames in PySpark?**

We can do it by making use of the createDataFrame() method of the SparkSession.

data = [('Harry', 20),

('Ron', 20),

('Hermoine', 20)]

columns = ["Name","Age"]

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

This creates the dataframe as shown below:

+-----------+----------+

| Name | Age |

+-----------+----------+

| Harry | 20 |

| Ron | 20 |

| Hermoine | 20 |

+-----------+----------+

We can get the schema of the dataframe by using df.printSchema()

>> df.printSchema()

root

|-- Name: string (nullable = true)

|-- Age: integer (nullable = true)

### **26. Is it possible to create PySpark DataFrame from external data sources?**

Yes, it is! Realtime applications make use of external file systems like local, HDFS, HBase, MySQL table, S3 Azure etc. Following example shows how we can create DataFrame by reading data from a csv file present in the local system:

df = spark.read.csv("/path/to/file.csv")

PySpark supports csv, text, avro, parquet, tsv and many other file extensions.

### **27. What do you understand by Pyspark’s startsWith() and endsWith() methods?**

These methods belong to the Column class and are used for searching DataFrame rows by checking if the column value starts with some value or ends with some value. They are used for filtering data in applications.

* ****startsWith()**** – returns boolean Boolean value. It is true when the value of the column starts with the specified string and False when the match is not satisfied in that column value.
* ****endsWith()**** – returns boolean Boolean value. It is true when the value of the column ends with the specified string and False when the match is not satisfied in that column value.

Both the methods are case-sensitive.

Consider an example of the startsWith() method here. We have created a DataFrame with 3 rows:

data = [('Harry', 20),

('Ron', 20),

('Hermoine', 20)]

columns = ["Name","Age"]

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

If we have the below code that checks for returning the rows where all the names in the Name column start with “H”,

**import** org.apache.spark.sql.functions.col

df.filter(col("Name").startsWith("H")).show()

The output of the code would be:

+-----------+----------+

| Name | Age |

+-----------+----------+

| Harry | 20 |

| Hermoine | 20 |

+-----------+----------+

Notice how the record with the Name “Ron” is filtered out because it does not start with “H”.

### **28. What is PySpark SQL?**

PySpark SQL is the most popular PySpark module that is used to process structured columnar data. Once a DataFrame is created, we can interact with data using the SQL syntax. Spark SQL is used for bringing native raw SQL queries on Spark by using select, where, group by, join, union etc. For using PySpark SQL, the first step is to create a temporary table on DataFrame by using createOrReplaceTempView() function. Post creation, the table is accessible throughout SparkSession by using sql() method. When the SparkSession gets terminated, the temporary table will be dropped.  
For example, consider we have the following DataFrame assigned to a variable df:

+-----------+----------+----------+

| Name | Age | Gender |

+-----------+----------+----------+

| Harry | 20 | M |

| Ron | 20 | M |

| Hermoine | 20 | F |

+-----------+----------+----------+

In the below piece of code, we will be creating a temporary table of the DataFrame that gets accessible in the SparkSession using the sql() method. The SQL queries can be run within the method.

df.createOrReplaceTempView("STUDENTS")

df\_new = spark.sql("SELECT \* from STUDENTS")

df\_new.printSchema()

The schema will be displayed as shown below:

>> df.printSchema()

root

|-- Name: string (nullable = true)

|-- Age: integer (nullable = true)

|-- Gender: string (nullable = true)

For the above example, let’s try running group by on the Gender column:

groupByGender = spark.sql("SELECT Gender, count(\*) as Gender\_Count from STUDENTS group by Gender")

groupByGender.show()

The above statements results in:

+------+------------+

|Gender|Gender\_Count|

+------+------------+

| F| 1 |

| M| 2 |

+------+------------+

### **29. How can you inner join two DataFrames?**

We can make use of the join() method present in PySpark SQL. The syntax for the function is:

join(self, other, on=None, how=None)

where,  
other - Right side of the join  
on - column name string used for joining  
how - type of join, by default it is inner. The values can be inner, left, right, cross, full, outer, left\_outer, right\_outer, left\_anti, left\_semi.

The join expression can be appended with where() and filter() methods for filtering rows. We can have multiple join too by means of the chaining join() method.

Consider we have two dataframes - employee and department as shown below:

-- Employee DataFrame --

+------+--------+-----------+

|emp\_id|emp\_name|empdept\_id |

+------+--------+-----------+

| 1| Harry| 5|

| 2| Ron | 5|

| 3| Neville| 10|

| 4| Malfoy| 20|

+------+--------+-----------+

-- Department DataFrame --

+-------+--------------------------+

|dept\_id| dept\_name |

+-------+--------------------------+

| 5 | Information Technology |

| 10| Engineering |

| 20| Marketting |

+-------+--------------------------+

We can inner join the Employee DataFrame with Department DataFrame to get the department information along with employee information as:

emp\_dept\_df = empDF.join(deptDF,empDF.empdept\_id == deptDF.dept\_id,"inner").show(truncate=False)

The result of this becomes:

+------+--------+-----------+-------+--------------------------+

|emp\_id|emp\_name|empdept\_id |dept\_id| dept\_name |

+------+--------+-----------+-------+--------------------------+

| 1| Harry| 5| 5 | Information Technology |

| 2| Ron | 5| 5 | Information Technology |

| 3| Neville| 10| 10 | Engineering |

| 4| Malfoy| 20| 20 | Marketting |

+------+--------+-----------+-------+--------------------------+

We can also perform joins by chaining join() method by following the syntax:

df1.join(df2,["column\_name"]).join(df3,df1["column\_name"] == df3["column\_name"]).show()

Consider we have a third dataframe called Address DataFrame having columns emp\_id, city and state where emp\_id acts as the foreign key equivalent of SQL to the Employee DataFrame as shown below:

-- Address DataFrame --

+------+--------------+------+

|emp\_id| city |state |

+------+--------------+------+

|1 | Bangalore | KA |

|2 | Pune | MH |

|3 | Mumbai | MH |

|4 | Chennai | TN |

+------+--------------+------+

If we want to get address details of the address along with the Employee and the Department Dataframe, then we can run,

resultDf = empDF.join(addressDF,["emp\_id"])

.join(deptDF,empDF["empdept\_id"] == deptDF["dept\_id"])

.show()

The resultDf would be:

+------+--------+-----------+--------------+------+-------+--------------------------+

|emp\_id|emp\_name|empdept\_id | city |state |dept\_id| dept\_name |

+------+--------+-----------+--------------+------+-------+--------------------------+

| 1| Harry| 5| Bangalore | KA | 5 | Information Technology |

| 2| Ron | 5| Pune | MH | 5 | Information Technology |

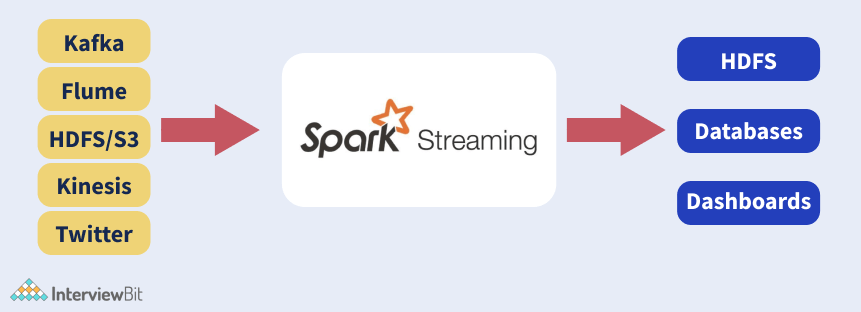
| 3| Neville| 10| Mumbai | MH | 10 | Engineering |

| 4| Malfoy| 20| Chennai | TN | 20 | Marketting |

+------+--------+-----------+--------------+------+-------+--------------------------+

### **30. What do you understand by Pyspark Streaming? How do you stream data using TCP/IP Protocol?**

PySpark Streaming is scalable, fault-tolerant, high throughput based processing streaming system that supports streaming as well as batch loads for supporting real-time data from data sources like TCP Socket, S3, Kafka, Twitter, file system folders etc. The processed data can be sent to live dashboards, Kafka, databases, HDFS etc.



To perform Streaming from the TCP socket, we can use the readStream.format("socket") method of Spark session object for reading data from TCP socket and providing the streaming source host and port as options as shown in the code below:

**from** pyspark **import** SparkContext**from** pyspark.streaming **import** StreamingContext**from** pyspark.sql **import** SQLContext**from** pyspark.sql.functions **import** desc

sc = SparkContext()

ssc = StreamingContext(sc, 10)

sqlContext = SQLContext(sc)

socket\_stream = ssc.socketTextStream("127.0.0.1", 5555)

lines = socket\_stream.window(20)

df.printSchema()

Spark loads the data from the socket and represents it in the value column of the DataFrame object. The df.printSchema() prints

root

|-- value: string (nullable = true)

Post data processing, the DataFrame can be streamed to the console or any other destinations based on the requirements like Kafka, dashboards, database etc.

### **31. What would happen if we lose RDD partitions due to the failure of the worker node?**

If any RDD partition is lost, then that partition can be recomputed using operations lineage from the original fault-tolerant dataset.

### 1) What is PySpark? / What do you know about PySpark?

PySpark is a tool or interface of Apache Spark developed by the Apache Spark community and Python to support Python to work with Spark. This tool collaborates with Apache Spark using APIs written in Python to support features like Spark SQL, Spark DataFrame, Spark Streaming, Spark Core, Spark MLlib, etc. It provides an interactive PySpark shell to analyze structured and semi-structured data in a distributed environment and process them by providing optimized APIs that help the program to read data from various data sources. PySpark features are implemented in the py4j library in Python. Due to the availability of the Py4j library, it facilitates users to work with RDDs (Resilient Distributed Datasets) in the Python programming language. Python supports many libraries that support big data processing and machine learning.

You can install PySpark using PyPi by using the following command:

1. pip install pyspark

### 2) What are the main characteristics of PySpark?

Following are the main four main characteristics of PySpark:

* ****Nodes are abstracted:**** The nodes are abstracted in PySpark. It means we cannot access the individual worker nodes.
* ****PySpark is based on MapReduce:**** PySpark is based on the MapReduce model of Hadoop. It means that the programmer provides the map and the reduced functions.
* ****APIs for Spark features:**** PySpark provides APIs for utilizing Spark features.
* ****Abstracted Network:**** PySpark provides abstracted networks. It means that the networks are abstracted in PySpark, and it facilitates only implicit communication.

### 3) What is RDD in PySpark?

In PySpark, RDD is an acronym that stands for Resilient Distributed Datasets. It is a core data structure of PySpark. It is a low-level object that is highly efficient in performing distributed tasks.

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner%26utm_campaign=growth_hack)

The PySpark's RDDs are the elements that can run and operate on multiple nodes to do parallel processing on a cluster. These are immutable elements. It means that if you once create an RDD, you cannot change it. RDDs are also fault-tolerant. In the case of any failure, they recover automatically. We can apply multiple operations on RDDs to achieve a certain task.

### 4) What are the key advantages and disadvantages of PySpark?

Following is a list of key advantages and disadvantages of PySpark:

****Advantages of PySpark****

* PySpark is an easy-to-learn language. You can learn and implement it easily if you know Python and Apache Spark.
* PySpark is simple to use. It provides parallelized codes that are simple to write.
* Error handling is simple in the PySpark framework. You can easily handle errors and manage synchronization points
* PySpark is a Python API for Apache Spark. It provides great library support. Python has a huge library collection for working in data science and data visualization compared to other languages.
* Many important algorithms are already written and implemented in Spark. It provides many algorithms in Machine Learning or Graphs.

****Disadvantages of PySpark****

* PySpark is based on Hadoop's MapReduce model, so sometimes, it becomes challenging to manage and express problems using the MapReduce model.
* Since Apache Spark was originally written in Scala while using PySpark in Python programs, they are not as efficient as other programming models. It is approximate 10x times slower than the Scala programs. Due to this reason, it negatively impacts the performance of heavy data processing applications.
* The Spark Streaming API in PySpark is not as efficient as Scala. It still requires improvements.
* In PySpark, the nodes are abstracted, and it uses the abstracted network, so it cannot be used to modify the internal function of the Spark. Scala is preferred in this case.

### 5) What are the prerequisites to learn PySpark?

PySpark is easy to learn and implement. It doesn't require the expertise of many programming languages or databases. You can learn it easily if you know a programming language and framework. Before learning the concept of PySpark, you should learn some knowledge of Apache Spark and Python. It will be very helpful to learn the advanced concepts of PySpark.

### 6) Why are Partitions immutable in PySpark?

In PySpark, every transformation generates a new partition. Partitions use HDFS API to make partitions immutable, distributed, and fault-tolerant. Partitions are also aware of data locality.

### 7) What are the key differences between an RDD, a DataFrame, and a DataSet?

Following are the key differences between an RDD, a DataFrame, and a DataSet:

****RDD****

* RDD is an acronym that stands for Resilient Distributed Dataset. It is a core data structure of PySpark.
* RDD is a low-level object that is highly efficient in performing distributed tasks.
* RDD is best to do low-level transformations, operations, and control on a dataset.
* RDD is mainly used to alter data with functional programming structures than with domain-specific expressions.
* If you have a similar arrangement of data that needs to be calculated again, RDDs can be efficiently reserved.
* RDD contains all datasets and DataFrames in PySpark.

****DataFrame****

* A DataFrame is equivalent to a relational table in Spark SQL. It facilitates the structure like lines and segments to be seen.
* If you are working on Python, it is best to start with DataFrames and then switch to RDDs if you want more flexibility.
* One of the biggest disadvantages of DataFrames is Compile Time Wellbeing. For example, if the information structure is unknown, you cannot control it.

****DataSet****

* A Dataset is a distributed collection of data. It is a subset of DataFrames.
* Dataset is a newly added interface in Spark 1.6 to provide RDD benefits.
* DataSet consists of the best encoding component. It provides time security in an organized manner, unlike information edges.
* DataSet provides a greater level of type safety at compile-time. It can be used if you want typed JVM objects.
* By using DataSet, you can take advantage of Catalyst optimization. You can also use it to benefit from Tungsten's fast code generation.

### 8) What do you understand by PySpark SparkContext?

SparkContext acts as the entry point to any spark functionality. When the Spark application runs, it starts the driver program, and the main function and SparkContext get initiated. After that, the driver program runs the operations inside the executors on worker nodes. In PySpark, SparkContext is known as PySpark SparkContext. It uses Py4J (library) to launch a JVM and then creates a JavaSparkContext. The PySpark's SparkContext is by default available as 'sc', so it doesn't mean creating a new SparkContext.

### 9) What is the usage of PySpark StorageLevel?

The PySpark StorageLevel is used to control the storage of RDD. It controls how and where the RDD is stored. PySpark StorageLevel decides if the RDD is stored on the memory, over the disk, or both. It also specifies whether we need to replicate the RDD partitions or serialize the RDD.

****Following is the code for PySpark StorageLevel:****

1. class pyspark.StorageLevel( useDisk, useMemory, useOfHeap, deserialized, replication = 1)

### 10) What do you understand by data cleaning?

Data cleaning is the process of preparing data by analyzing the data and removing or modifying data if it is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

### 11) What is PySpark SparkConf?

PySpark SparkConf is mainly used if we have to set a few configurations and parameters to run a Spark application on the local/cluster. In other words, we can say that PySpark SparkConf is used to provide configurations to run a Spark application.

### 12) What are the different types of algorithms supported in PySpark?

Different types of algorithms supported in PySpark are:

* spark.mllib
* mllib.regression
* mllib.recommendation
* mllib.clustering
* mllib.classification
* mllib.linalg
* mllib.fpm

### 13) What is SparkCore, and what are the key functions of SparkCore?

SparkCore is a general execution engine for the Spark platform, including all the functionalities. It offers in-memory computing capabilities to deliver a good speed, a generalized execution model to support various applications, and Java, Scala, and Python APIs that make the development easy.

The main responsibility of SparkCore is to perform all the basic I/O functions, scheduling, monitoring, etc. It is also responsible for fault recovery and effective memory management.

****The key functions of SparkCore are:****

* Perform all the basic I/O functions
* Job scheduling
* Monitoring jobs
* Memory management
* Fault-tolerance
* Interaction with storage systems

#### Note: It also includes additional libraries that can divide the workloads for streaming, machine learning, and SQL.

### 14) What do you know about PySpark SparkFiles?

PySpark facilitates users to upload their files using sc.addFile. Here, sc is our default SparkContext. We can also get the path of the working directory using SparkFiles.get. SparkFiles provides the following types of class methods to resolve the path to the files added through SparkContext.addFile():

* get(filename)
* getrootdirectory()

### 15) What do you know about PySpark serializers?

In PySpark, serialization is a process that is used to conduct performance tuning on Spark. PySpark supports serializers because we have to continuously check the data sent or received over the network to the disk or memory. PySpark supports two types of serializers. They are as follows:

* ****PickleSerializer:**** This is used to serialize the objects using Python's PickleSerializer using class pyspark.PickleSerializer). This serializer supports almost every Python object.
* ****MarshalSerializer:**** The MarshalSerializer is used to perform serialization of objects. This can be used by using class pyspark.MarshalSerializer. This serializer is way faster than the PickleSerializer, but it supports only limited types.

### 16) What is PySpark ArrayType? Give an example to explain it well.

PySpark ArrayType is a collection data type that extends the PySpark's DataType class, which is the superclass for all kinds. The PySpark ArrayType contains only the same types of items. The ArraType() method can also be used to construct an instance of an ArrayType.

It accepts two arguments:

* ****valueType:**** The valueType should extend the DataType class in PySpark.
* ****valueContainsNull:**** It is an optional argument. It specifies whether a value can accept null and is set to True by default.

****Example:****

1. from pyspark.sql.types import StringType, ArrayType
2. arrayCol = ArrayType(StringType(),False)

### 17) What are the most frequently used Spark ecosystems?

The most frequently used Spark ecosystems are:

* Spark SQL for developers. It is also known as Shark.
* Spark Streaming for processing live data streams.
* Graphx for generating and computing graphs.
* MLlib (also known as Machine Learning Algorithms)
* SparkR to promote R programming language in Spark engine.

### 18) What machine learning API does PySpark provide?

Just like Apache Spark, PySpark also provides a machine learning API known as MLlib. MLlib supports the following types of machine learning algorithms:

* ****mllib.classification:**** This machine learning API supports different methods for binary or multiclass classification and regression analysis such as Random Forest, Decision Tree, Naive Bayes, etc.
* ****mllib.clustering:**** This machine learning API solves clustering problems for grouping entities subsets with one another depending on similarity.
* ****mllib.fpm:**** FPM stands for Frequent Pattern Matching in this machine learning API. This machine learning API is used to mine frequent items, subsequences, or other structures that are used for analyzing large datasets.
* ****mllib.linalg:**** This machine learning API is used to solve problems on linear algebra.
* ****mllib.recommendation:**** This machine learning API is used for collaborative filtering and recommender systems.
* ****spark.mllib:**** This machine learning API is used to support model-based collaborative filtering where small latent factors are identified using the Alternating Least Squares (ALS) algorithm used for predicting missing entries.
* ****mllib.regression:**** This machine learning API solves problems by using regression algorithms that find relationships and variable dependencies.

### 19) What is PySpark Partition? How many partitions can you make in PySpark?

PySpark Partition is a method of splitting a large dataset into smaller datasets based on one or more partition keys. It enhances the execution speed as transformations on partitioned data run quicker because each partition's transformations are executed in parallel. PySpark supports both partitioning in memory (DataFrame) and partitioning on disc (File system). When we make a DataFrame from a file or table, PySpark creates the DataFrame in memory with a specific number of divisions based on specified criteria.

It also facilitates us to create a partition on multiple columns using partitionBy() by passing the columns you want to partition as an argument to this method.

****Syntax:****

1. partitionBy(self, \*cols)

In PySpark, it is recommended to have 4x of partitions to the number of cores in the cluster available for application.

### 20) What do you understand by PySpark DataFrames?

PySpark DataFrames are the distributed collection of well-organized data. These are the same as relational databases tables and are placed into named columns. PySpark DataFrames are better optimized than R or Python programming language because these can be created from different sources like Hive Tables, Structured Data Files, existing RDDs, external databases, etc.

The biggest advantage of PySpark DataFrame is that the data in the PySpark DataFrame is distributed across different machines in the cluster, and the operations performed on this would be run parallel on all the machines. This facilitates handling a large collection of structured or semi-structured data of a range of petabytes.

### 21) What do you understand by "joins" in PySpark DataFrame? What are the different types of joins available in PySpark?

In PySpark, joins merge or join two DataFrames together. It facilitates us to link two or multiple DataFrames together.

INNER Join, LEFT OUTER Join, RIGHT OUTER Join, LEFT ANTI Join, LEFT SEMI Join, CROSS Join, and SELF Join are among the SQL join types PySpark supports. Following is the syntax of PySpark Join.

****Syntax:****

1. join(self, other, on=None, how=None)

****Parameter Explanation:****

The join() procedure accepts the following parameters and returns a DataFrame:

* ****"other":**** It specifies the join's right side.
* ****"on":**** It specifies the join column's name.
* ****"how":**** It is used to specify an option. Options are inner, cross, outer, full, full outer, left, left outer, right, right outer, left semi, and left anti. The default is inner.

****Types of Join in PySpark DataFrame****

|  |  |
| --- | --- |
| **Join String** | **Equivalent SQL Join** |
| inner | INNER JOIN |
| outer, full, fullouter, full\_outer | FULL OUTER JOIN |
| left, leftouter, left\_outer | LEFT JOIN |
| right, rightouter, right\_outer | RIGHT JOIN |
| cross |  |
| anti, leftanti, left\_anti |  |
| semi, leftsemi, left\_semi |  |

### 22) What is Parquet file in PySpark?

In PySpark, the Parquet file is a column-type format supported by several data processing systems. By using the Parquet file, Spark SQL can perform both read and write operations.

The Parquet file contains a column type format storage which provides the following advantages:

* It is small and consumes less space.
* It facilitates us to fetch specific columns for access.
* It follows type-specific encoding.
* It offers better-summarized data.
* It contains very limited I/O operations.

### 23) What do you understand by a cluster manager? What are the different cluster manager types supported by PySpark?

In PySpark, a cluster manager is a cluster mode platform that facilitates Spark to run by providing all resources to worker nodes according to their requirements.

A Spark cluster manager ecosystem contains a master node and multiple worker nodes. The master nodes provide the worker nodes with the resources like memory, processor allocation, etc., according to the nodes' requirements with the help of the cluster manager.

****PySpark supports the following cluster manager types:****

* ****Standalone:**** This is a simple cluster manager that comes with Spark.
* ****Apache Mesos:**** This cluster manager is used to run Hadoop MapReduce and PySpark apps.
* ****Hadoop YARN:**** This cluster manager is used in Hadoop2.
* ****Kubernetes:**** This cluster manager is an open-source cluster manager that helps automate deployment, scaling, and automatic management of containerized apps.
* ****local:**** This cluster manager is a mode for running Spark applications on laptops/desktops.

### 24) Why is PySpark faster than pandas?

PySpark is faster than pandas because it supports the parallel execution of statements in a distributed environment. For example, PySpark can be executed on different cores and machines, unavailable in Pandas. This is the main reason why PySpark is faster than pandas.

### 25) What is the difference between get(filename) and getrootdirectory()?

The main difference between get(filename) and getrootdirectory() is that the get(filename) is used to achieve the correct path of the file that is added through SparkContext.addFile(). On the other hand, the getrootdirectory() is used to get the root directory containing the file added through SparkContext.addFile().

### 26) What do you understand by SparkSession in Pyspark?

In PySpark, SparkSession is the entry point to the application. In the first version of PySpark, SparkContext was used as the entry point. SparkSession is the replacement of SparkContext since PySpark version 2.0. After the PySpark version 2.0, SparkSession acts as a starting point to access all of the PySpark functionalities related to RDDs, DataFrame, Datasets, etc. It is also a Unified API used to replace the SQLContext, StreamingContext, HiveContext, and all other contexts in Pyspark.

The SparkSession internally creates SparkContext and SparkConfig according to the details provided in SparkSession. You can create SparkSession by using builder patterns.

### 27) What are the key advantages of PySpark RDD?

Following is the list of key advantages of PySpark RDD:

****Immutability:**** The PySpark RDDs are immutable. If you create them once, you cannot modify them later. You have to create a new RDD whenever you try to apply any transformation operations on the RDDs.

****Fault Tolerance:**** The PySpark RDD provides fault tolerance features. Whenever an operation fails, the data gets automatically reloaded from other available partitions. This provides a seamless experience of execution of the PySpark applications.

****Partitioning:**** When we create an RDD from any data, the elements in the RDD are partitioned to the cores available by default.

****Lazy Evolution:**** PySpark RDD follows the lazy evolution process. In PySpark RDD, the transformation operations are not performed as soon as they are encountered. The operations would be stored in the DAG and are evaluated once it finds the first RDD action.

****In-Memory Processing:**** The PySpark RDD is used to help in loading data from the disk to the memory. You can persist RDDs in the memory for reusing the computations.

### 28) Explain the common workflow of a spark program.

The common workflow of a spark program can be described in the following steps:

* In the first step, we create the input RDDs depending on the external data. Data can be obtained from different data sources.
* After creating the PySpark RDDs, we run the RDD transformation operations such as filter() or map() to create new RDDs depending on the business logic.
* If we require any intermediate RDDs to reuse for later purposes, we can persist those RDDs.
* Finally, if any action operations like first(), count(), etc., are present, Spark launches it to initiate parallel computation.

### 29) How can you implement machine learning in Spark?

We can implement machine learning in Spark by using MLlib. Spark provides a scalable machine learning record called MLlib. It is mainly used to create machine learning scalable and straightforward with ordinary learning algorithms and use cases like clustering, weakening filtering, dimensional lessening, etc.

### 30) What do you understand by custom profilers in PySpark?

PySpark supports custom profilers. The custom profilers are used for building predictive models. Profilers are also used for data review to ensure that it is valid, and we can use it in consumption. When we require a custom profiler, it has to define some of the following methods:

* ****stats:**** This is used to return collected stats of profiling.
* ****profile:**** This is used to produce a system profile of some sort.
* ****dump:**** This is used to dump the profiles to a specified path.
* ****dump(id, path):**** This is used to dump a specific RDD id to the path given.
* ****add:**** This is used for adding profile to existing accumulated profile. The profile class has to be selected at the time of SparkContext creation.

### 31) What do you understand by Spark driver?

The Spark driver is a plan that runs on the master node of a machine. It is mainly used to state actions and alterations on data RDDs.

### 32) What is PySpark SparkJobinfo?

The PySpark SparkJobinfo is used to get information about the SparkJobs that are in execution.

Following is the code for using the SparkJobInfo:

1. class SparkJobInfo(namedtuple("SparkJobInfo", "jobId stageIds status ")):

### 33) What are the main functions of Spark core?

The main task of Spark Core is to implement several vital functions such as memory management, fault-tolerance, monitoring jobs, job setting up, and communication with storage systems. It also contains additional libraries, built atop the middle that is used to diverse workloads for streaming, machine learning, and SQL.

****The Spark Core is mainly used for the following tasks:****

* Fault tolerance and recovery.
* To interact with storage systems.
* Memory management.
* Scheduling and monitoring jobs on a cluster.

### 34) What do you understand by PySpark SparkStageinfo?

The PySpark SparkStageInfo is used to get information about the SparkStages available at that time. Following is the code used for SparkStageInfo:

1. class SparkStageInfo(namedtuple("SparkStageInfo", "stageId currentAttemptId name numTasks unumActiveTasks" "numCompletedTasks numFailedTasks" )):

### 35) What is the use of Spark execution engine?

The Apache Spark execution engine is a chart execution engine that facilitates users to examine massive data sets with a high presentation. You need to detain Spark in the memory to pick up performance radically if you want data to be manipulated with manifold stages of processing.

### 36) What is the use of Akka in PySpark?

Akka is used in PySpark for scheduling. When a worker requests a task to the master after registering, the master assigns a task to him. In this case, Akka sends and receives messages between the workers and masters.

### 37) What do you understand by startsWith() and endsWith() methods in PySpark?

The startsWith() and endsWith() methods in PySpark belong to the Column class and are used to search DataFrame rows by checking if the column value starts with some value or ends with some value. Both are used for filtering data in applications.

* ****startsWith() method:**** This method is used to return a Boolean value. It shows TRUE when the column's value starts with the specified string and FALSE when the match is not satisfied in that column value.
* ****endsWith() method:**** This method is used to return a Boolean value. It shows TRUE when the column's value ends with the specified string and FALSE when the match is not satisfied in that column value. Both methods are case-sensitive.

### 38) What do you understand by RDD Lineage?

The RDD lineage is a procedure that is used to reconstruct the lost data partitions. The Spark does not hold up data replication in the memory. If any data is lost, we have to rebuild it using RDD lineage. This is the best use case as RDD always remembers how to construct from other datasets.

### 39) Can we create PySpark DataFrame from external data sources?

Yes, we can create PySpark DataFrame from external data sources. The real-time applications use external file systems like local, HDFS, HBase, MySQL table, S3 Azure, etc. The following example shows how to create DataFrame by reading data from a csv file present in the local system:

1. df = spark.read.csv("/path/to/file.csv")

PySpark supports csv, text, avro, parquet, tsv and many other file extensions.

### 40) What are the main attributes used in SparkConf?

Following is the list of main attributes used in SparkConf:

* ****set(key, value):**** This attribute is used for setting the configuration property.
* ****setSparkHome(value):**** This attribute enables the setting Spark installation path on worker nodes.
* ****setAppName(value):**** This attribute is used for setting the application name.
* ****setMaster(value):**** This attribute is used to set the master URL.
* ****get(key, defaultValue=None):**** This attribute supports getting a configuration value of a key.

li>

### 41) How can you associate Spark with Apache Mesos?

We can use the following steps to associate Spark with Mesos:

* First, configure the sparkle driver program to associate with Mesos.
* The Spark paired bundle must be in the area open by Mesos.
* After that, install Apache Spark in a similar area as Apache Mesos and design the property "spark.mesos.executor.home" to point to the area where it is introduced.

### 42) What are the main file systems supported by Spark?

Spark supports the following three file systems:

* Local File system.
* Hadoop Distributed File System (HDFS).
* Amazon S3

### 43) How can we trigger automatic cleanups in Spark to handle accumulated metadata?

We can trigger the automatic cleanups in Spark by setting the parameter ' Spark.cleaner.ttl' or separating the long-running jobs into dissimilar batches and writing the mediator results to the disk.

### 44) How can you limit information moves when working with Spark?

We can limit the information moves when working with Spark by using the following manners:

* Communicate
* Accumulator factors

### 45) How is Spark SQL different from HQL and SQL?

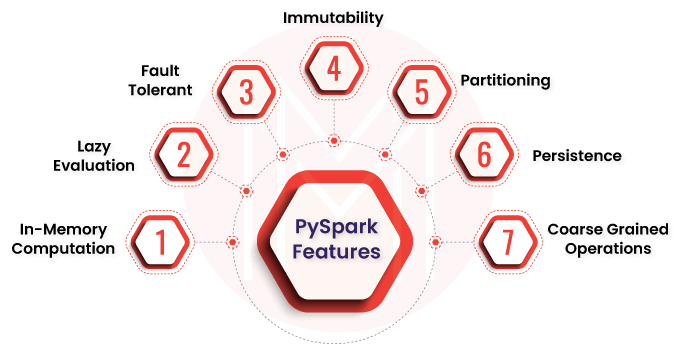
Hive is used in HQL (Hive Query Language), and Spark SQL is used in Structured Query language for processing and querying data. We can easily join SQL table and HQL table to Spark SQL. Flash SQL is used as a unique segment on the Spark Core motor that supports SQL and Hive Query Language without changing any sentence structure.

### 46) What is DStream in PySpark?

In PySpark, DStream stands for Discretized Stream. It is a group of information or gathering of RDDs separated into little clusters. It is also known as Apache Spark Discretized Stream and is used as a gathering of RDDs in the grouping. DStreams are based on Spark RDDs and are used to enable Streaming to flawlessly coordinate with some other Apache Spark segments like Spark MLlib and Spark SQL.

### **1. Explain PySpark.**

PySpark is a software-based on a python programming language with an inbuilt API. It was developed in Scala and released by the Spark community. It supports the Data Science team in working with Big Data. PySpark is a good learn for doing more scalability in analysis and data science pipelines.



### **2. Tell me the differences between PySpark and other programming languages.**

* It has an inbuilt API, whereas, in other programming languages, we need to integrate API externally from a third party.
* Implicit communication can be done in PySpark, but it is impossible in other programming languages.
* Developers can use the map to reduce function as PySpark is map-based.
* We can address multiple nodes in PySpark, which is impossible in other programming languages.

### **3. Why should we use PySpark?**

* Due to the most helpful ML algorithms implemented in PySpark, we can use it in Data Science.
* We can manage synchronization points and errors.
* Easy problems can be resolved quickly because all code is parallelized.

### **4. What are the main characteristics of PySpark?**

The primary characteristics of PySpark are listed below:

* ****Nodes are abstracted -**** This means we can’t address an individual node.
* ****Network is abstracted -**** Only implicit communication is possible here.
* ****Based on Map-Reduce -**** Additionally, programmers provide a reduce and map function.
* ****API for Spark -**** PySPark is a Spark API.

|  |
| --- |
| Looking forward to a career in a**Big Data Analytics**? Check out the "**[PySpark Training](https://mindmajix.com/pyspark-training" \o "PySpark Training" \t "https://mindmajix.com/_blank)**" and get certified today |

### **5. What are the advantages of PySpark?**

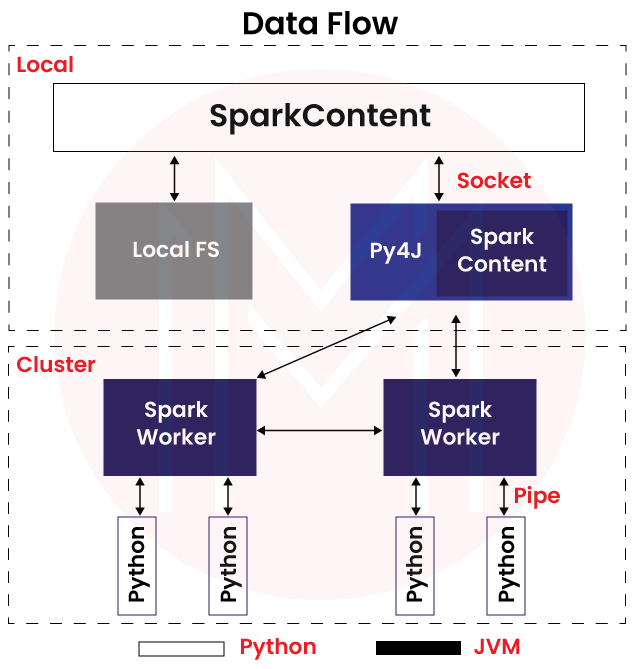
* ****Easy to write -**** For simple problems, it’s easy to write parallelized codes.
* ****Error handling -****Framework easily handles errors when it comes to synchronization points.
* ****Algorithms -****Most of the algorithms are already implemented in Spark.
* ****In-memory computation -****Through in-memory processing, Spark helps you to increase the processing speed. And the best thing is data is cached, thus allowing to fetch data from the disk every time, saving time.
* ****Swift processing -****One of the significant benefits of working with Spark is it provides a high processing speed of 10x faster on the disk and 100x faster in memory.
* ****Fault-tolerance -****Spark is specially designed to manage the malfunction of any worker node in the cluster, assuring that data loss is decreased to zero.

### **6. Tell me the disadvantages of PySpark.**

* While using the Mapreduce process, we may face some errors.
* It is more efficient for a significant amount of data, so we can face less accuracy when dealing with a small data set.

## **7. What do you mean by SparkContext?**

SparkContext is the software entry point for PySpark developers. When the developers try to launch this software, CparkContext will launch JVM using Py4J ( One of Python Library). This is a default process to provide as'sc' to the PySpark API.



### **8. Explain SparkConf and how does it work?**

Once the developer wants to run the Spark API locally in a cluster, they need to use SparkConf to configure the declared data parameters. We can write conf=new SparkConf().setMaster(local[2]) to declare the particular parameters.



### **9. What do you know about SparkFiles?**

To get the actual path of a file inside Apache Spark, we need to use **[SparkFiles](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.SparkFiles.html" \o "SparkFiles" \t "https://mindmajix.com/_blank)**. This is one of the Spark objects and can be added through SparkConf. We can access Spark jobs using SparkFiles. We can get the directory path through SparkFiles. We can set the recursive value to true so that directory will open.

[](https://bit.ly/3if9dmk)

### **10. Why do we need to mention the filename?**

Developers can find out the files by their filenames as the file extension is attached... Developers can understand file names by the filename first portion. For say, "setup" is the first part of setupact.log, so the file name is a setup that developers can understand easily.

### **11. Describe getrootdirectory ().**

The developers can obtain the root directory by using getrootdirectory().

It assists in obtaining the root directory, which contains the files added using SparkContext.addFile().

### **12. What is PySpark Storage Level?**

Storage level defines how RDD( Resilient Distributed Dataset) will be stored in a database. It also determines the storage capacity and focuses on data serialization.

### **13. Explain broadcast variables in PySpark.**

Developers can save the data as a copy into all nodes. All the data are variable fetched from machines and not sent back to devices. Broadcast variables will do code block to save the data copy as one of the classes of PySpark.

### **14. Why does the developer needs to do Serializers in PySpark?**

We can manage the data by serializers to tune the process. cPickle serializers are most effective for Python PySpark. It can handle any Python object. There are other serializers like Marshal, which doesn't support all Python objects.

### **15. When do you use Spark Stage info?**

In PySpark, developers can see the information about the Spark stages by using spark stage info. This is a physical unit that executes multiple tasks in computation. Spark stage info is controlled by DAG(Directed Acyclic Graph to process and transform any data.

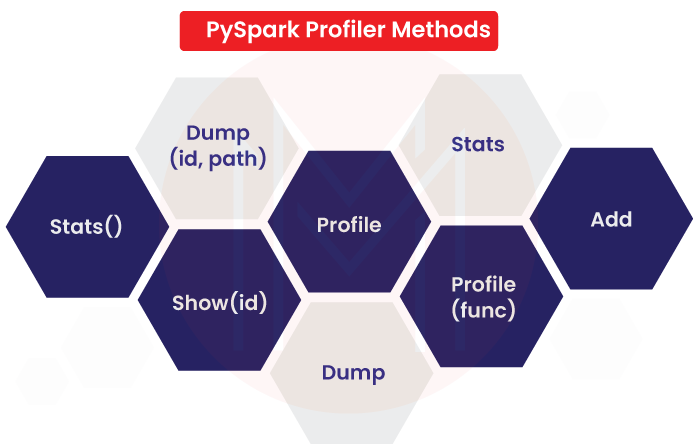
### **16. Which specific profiler do we use in PySpark?**

Only one profiler is supported in PySpark and manages the usages of the custom profiler data. That means we can configure another profiler to maintain the output. We need to also declare the required methods for custom profilers :

* ****Add:****we can add another profiler or add to an existing profile. SparkContext build-up usually initiates the different profile classes.
* ****Dump:**** To dump all the profiles to a particular path, we need to use the Dump profiler.
* ****Stats:**** We can get back the gathered stats by using this stats profiler.
* ****Profile:****We need to use this to create a system profile as a defined object.

### **17. How would you like to use Basic Profiler?**

By default, this is the standard profiler. We can use this while doing conjunction in cProfile and the accumulator.



### **18. Can we use PySpark in the small data set?**

We should not use PySaprk in the small data set. It will not help us so much because it's typical library systems that have more complex objects than more accessible. It's best for the massive amount of data set.

### **19. What is PySpark Partition?**

PySpark Partition allows you to split a large dataset into smaller ones using one or more partition keys. You can also use partitionBy() to create a partition on multiple columns by simply passing columns you want to partition as an argument.

****Syntax:****

Syntax: partitionBy(self, \*cols)

### **20. How many partitions can you make in PySpark?**

PySpark/Spark creates a task for each partition. You can transfer data from one partition to another using Spark Shuffle operations. By default, 200 partitions are created by DataFrame shuffle operations.

## **PySpark interview questions and Answers for experienced:**

### **1. Tell me a few algorithms which support PySpark.**

There few algorithms which we can use in PySpark:

* mllib. classification
* mllib. clustering
* smllib.fpm
* mllib. linalg
* smllib. recommendation
* spark. Mllib
* Mllib. Regression

### **2. Tell me the different SparkContext parameters.**

Please find out the different SparkContext parameters:

* The cluster's master URL from which it connects.
* Our job's name is appName.
* Py files These are the.zip or.py files that need to be sent to the cluster and added to the PYTHIONPATH.
* Variables in the context of worker nodes.
* RDD serializer is a serializer for RDD data.
* Conf is an object of LSparkConf that allows you to set all of the Spark properties.
* JSC is a joint-stock company. It's a JavaSparkContext object.

### **3. What is RDD? How many types of RDDs are in PySpark?**

The complete form of RDD is Resilient Distributed Datasets which are the elements used to run and operate on multiple nodes simultaneously on the same cluster. It can perform parallel processing as they use immutable characteristics. Once developers create an RDD, they can not change it anymore. Once any failure happens, this RDD will be recovered automatically.

There are two types of RDD:

* ****Transformation:****This type of RDD is applicable in creating a new RDD or transforming any filter or map.
* ****Action:****This type of RDD performs some computations on the return values. It sends data from the executor to the driver.

### **4. Tell me the different cluster manager types in PySpark.**

There are many types of the cluster, few of them are:

* ****Local:****It simplifies the running mode for Spark application through API.
* ****Kubernetes:**** It helps in automated deployment and data scaling as an open-source cluster.
* ****Hadoop YARN:**** This type of cluster manages the Hadoop environment.
* ****Apache Mesos:**** In this cluster, we can run Map-reduce.
* ****Standalone:**** This cluster can operate the Spark API.

### **5. What do you understand about PySpark DataFrames?**

DataFrames can create Hive tables, structured data files, or RDD in PySpark. As PySpark is based on the rational database, this DataFrames organized data in equivalent tables and placed them in named columns. As a result, it has better optimization to compare the data set.

### **6. Explain SparkSession in PySpark.**

We use usually get entry in PySpark through SparkContext in version 2.0. But from version 3.0, we can get into it by using SparkSession. It acts as the starting point to access all PySpark functionalities like RDD or DataFrames. We can also use this to unified API.

### **7. What do you know about PySpark UDF?**

The complete form of UDF is User Defined Functions. It will be created when no functionalities do not support the PySpark library. Developers can create UDF by using the Python function and wrapping. SQL or DataFrames can reject it.

### **8. Describe PySpark Architecture.**

This architecture is mainly based on mater slave pattern. Here driver means master node, and worker means slave nodes. Worker nodes are the main operational point. The cluster manager can manage the whole operation on the worker nodes.

### **9. What do you know about the PySpark DAGScheduler?**

The complete form of DAG is Direct Acyclic Graph. It controls the scheduling layer of Spark for executing the stage-oriented scheduled tasks. This scheduler executes stages DAG for each job. Developers can keep track of all stages in RDD. Even this DAG scheduler reduces the running time.

### **10. Which workflow do we need to follow in PySpark?**

The typical workflows are:

* We need to create input RDD on the external data. These data can be taken from another source.
* Intermediate RDD needs to be created for later purposes.
* parallel computation is present in this workflow.

### **11. Tell me how RDD is created in PySpark?**

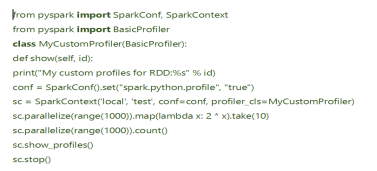
* ****Apply sparkContext.parallelize():**** Method parallelize() of SparkContext is used to create RDD by loading existing collection from the driver and then parallelizing. If the data is present in memory, we can only use this process.
* ****Apply sparkContext.textFile():**** If we are going to read from the text file and transfer them into RDD, we can use this method.
* ****Apply sparkContext.wholeTextFiles():**** The value of the file and file path can find out by using this method.
* ****Empty RDD with no partition using sparkContext.emptyRDD:****one empty RDD can be created by the method.
* ****Empty RDD with partitions using sparkContext.parallelize:****New empty RDD can be created without data in the partition.

### **12. Can we create a Data frame using an external database?**

We can create the data frame locally in HDFC, HBase, MySQL, and any cloud.

|  |
| --- |
| Check Out: **[Steps To Set-Up Your MySQL Reporting](https://mindmajix.com/the-4-steps-to-set-up-your-mysql-reporting" \o "Steps To Set-Up Your MySQL Reporting" \t "https://mindmajix.com/_blank)** |

### **13. Explain Add method.**



### **14. What is PySpark SQL?**

Spark SQL is a module in Spark for structured data processing. It offers DataFrames and also operates as a distributed SQL query engine. PySpark SQL may also read data from existing Hive installations. Further, data extraction is possible using an SQL query language.

### **15. Do you think PySpark is similar to SQL?**

In SQL database is maintained in tabular form. As well as in PySpark API, all information is stored in Data Frames. This Data Frame is immutable and stored in columns. That's why this is similar to SQL.

### **16. Why use Akka in PySpark?**

Spark makes use of Akka for scheduling primarily. After registration, all workers request a task to complete. The master simply assigns the work. Spark uses Akka to communicate between workers and masters in this case.

### **17. How is PySpark exposed in Big Data?**

The PySpark API is attached with the Spark programming model to Python and Apache Spark. Apache Spark is open-source software, so the most popular Big Data framework can scale up the process in a cluster and make it faster. Big Data use distributed database system in-memory data structures to smoother the processing.

## **Top 10 Frequently Asked Questions in an interview on PySpark:**

### **1. Why do we use PySpark?**

Python and its set of libraries in real-time for large-scale data. It can be used through an open-source Apache Spark. Software industries are using this PySpark as Python API.

### **2. Do you think that PySpark and Python are similar?**

Yes, they are directly related. It is a Python-based API that is based on the Spark framework. As a programming language, Python helps Spark manage big data.

### **3. Can we use PySpark as a programming language?**

No, we can not use PySpark as a programming language. It's a computing framework.

### **4. Which one is the faster, PySpark or Pandas?**

The processing speed depends upon the platform we are using to manage the vast amount of data. As PySpark is easy to use through inbuilt API, as a result, speed is faster. However, at the same time, Pandas is not running with any API; as a result, the rate is slower than PySpark.

### **5. Why is PySpark helpful for machine learning?**

As PySpark is working with Machine Learning on a distributed database system so they can work together efficiently. We can use PySpark in extensive data analysis by using ML and Python. It also runs smoothly with Tableau. Moreover, we can run different machine learning algorithms due to the PySpark ML library.

### **6. What do you think PySpark is important in Data Science?**

Data Science is based on two programming languages like Python and ML. PySpark is built into Python. It has the interface and inbuilt environment to use Python and ML both. That's why PySpark is an essential tool in Data Science. Once we process the data set, prototype models will be converted into production-grade workflows.

### **7. Name a few of the companies that are using PySpark?**

Most of the E-commerce industry, Banking Industry, IT Industry, Retail industry, etc., are using PySpark. A few of the companies' names are Trivago, Amazon, Walmart, Runtastic, Sanofi, etc.

### **8. What are the different MLlib tools available in Spark?**

MLlib can perform machine learning in Apache Spark. The different MLlib tools available in Spark are listed below:

* ****ML Algorithms:**** The core of MLlib is ML Algorithms. These include common learning algorithms like classification, clustering, regression, and collaborative filtering.
* ****Featurization:**** It includes extraction, transformation, selection, and dimensionality reduction.
* ****Pipelines:**** Pipelines provide tools to construct, evaluate, and tune ML Pipelines.
* ****Persistence:****It aids in saving and loading models, algorithms, and pipelines.
* ****Utilities:****Utilities for statistics, algebra, and handling data.

|  |
| --- |
| ****Check out:**[Machine Learning Tutorial](https://mindmajix.com/machine-learning-tutorial" \l "learning-tools" \o " Machine Learning Tutorial" \t "https://mindmajix.com/_blank)** |

### **9. Explain the function of SparkCore.**

SparkCore is the base engine for distributed data processing and large-scale parallel computation. SparkCore performs vital functions like memory management, fault-tolerance, job scheduling and monitoring, and interaction with storage systems. Furthermore, additional libraries built at the top of the core allow diverse SQL and machine learning workloads.

### **10. List the main attributes used in SparkConf.**

Below-listed are the most commonly used attributes of SparkConf:

* ****set(key, value) -**** It sets the configuration property.
* ****setMaster(value) -**** It sets the master URL.
* ****setAppName(value) -**** It sets the application name.
* ****get(key, defaultValue=None) -**** It gets the configuration value of a key.
* ****setSparkHome(value)****

#### Q1. What's the difference between an RDD, a DataFrame, and a DataSet?

RDD-

It is Spark's structural square. [RDDs](https://www.projectpro.io/article/apache-spark-ecosystem-and-spark-components/219" \o "RDDs" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) contain all datasets and dataframes.

If a similar arrangement of data needs to be calculated again, RDDs can be efficiently reserved.

It's useful when you need to do low-level transformations, operations, and control on a dataset.

It's more commonly used to alter data with functional programming structures than with domain-specific expressions.

DataFrame-

It allows the structure, i.e., lines and segments, to be seen. You can think of it as a database table.

Optimized Execution Plan- The catalyst analyzer is used to create query plans.

One of the limitations of dataframes is Compile Time Wellbeing, i.e., when the structure of information is unknown, no control of information is possible.

Also, if you're working on Python, start with DataFrames and then switch to RDDs if you need more flexibility.

DataSet (A subset of DataFrames)-

It has the best encoding component and, unlike information edges, it enables time security in an organized manner.

If you want a greater level of type safety at compile-time, or if you want typed JVM objects, Dataset is the way to go.

Also, you can leverage datasets in situations where you are looking for a chance to take advantage of Catalyst optimization or even when you are trying to benefit from Tungsten’s fast code generation.

**New Projects**

**[Learn to Build a Siamese Neural Network for Image Similarity](https://www.projectpro.io/project-use-case/siamese-neural-network-code-for-image-similarity?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/siamese-neural-network-code-for-image-similarity?utm_source=520&utm_medium=fold2)

**[Getting Started with Pyspark on AWS EMR and Athena](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)

**[Build Piecewise and Spline Regression Models in Python](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)

**[Getting Started with Pyspark on AWS EMR and Athena](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)

**[PyTorch Project to Build a GAN Model on MNIST Dataset](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)

**[Build a Text Generator Model using Amazon SageMaker](https://www.projectpro.io/project-use-case/text-generator-model-lstm-using-amazon-sagemaker?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/text-generator-model-lstm-using-amazon-sagemaker?utm_source=520&utm_medium=fold2)

**[PyTorch Project to Build a GAN Model on MNIST Dataset](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)

**[Build Real Estate Price Prediction Model with NLP and FastAPI](https://www.projectpro.io/project-use-case/real-estate-price-prediction-model-using-fastapi-and-heroku?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/real-estate-price-prediction-model-using-fastapi-and-heroku?utm_source=520&utm_medium=fold2)

**[Build Piecewise and Spline Regression Models in Python](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)

**[CycleGAN Implementation for Image-To-Image Translation](https://www.projectpro.io/project-use-case/cyclegan-implementation-pytorch-for-image-to-image-translation?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/cyclegan-implementation-pytorch-for-image-to-image-translation?utm_source=520&utm_medium=fold2)

**[Learn to Build a Siamese Neural Network for Image Similarity](https://www.projectpro.io/project-use-case/siamese-neural-network-code-for-image-similarity?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/siamese-neural-network-code-for-image-similarity?utm_source=520&utm_medium=fold2)

**[Getting Started with Pyspark on AWS EMR and Athena](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)

**[Build Piecewise and Spline Regression Models in Python](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)

**[Getting Started with Pyspark on AWS EMR and Athena](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/getting-started-with-pyspark-on-aws-emr-and-athena?utm_source=520&utm_medium=fold2)

**[PyTorch Project to Build a GAN Model on MNIST Dataset](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)

**[Build a Text Generator Model using Amazon SageMaker](https://www.projectpro.io/project-use-case/text-generator-model-lstm-using-amazon-sagemaker?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/text-generator-model-lstm-using-amazon-sagemaker?utm_source=520&utm_medium=fold2)

**[PyTorch Project to Build a GAN Model on MNIST Dataset](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/gan-mnist-pytorch?utm_source=520&utm_medium=fold2)

**[Build Real Estate Price Prediction Model with NLP and FastAPI](https://www.projectpro.io/project-use-case/real-estate-price-prediction-model-using-fastapi-and-heroku?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/real-estate-price-prediction-model-using-fastapi-and-heroku?utm_source=520&utm_medium=fold2)

**[Build Piecewise and Spline Regression Models in Python](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/piecewise-spline-regression-project?utm_source=520&utm_medium=fold2)

**[CycleGAN Implementation for Image-To-Image Translation](https://www.projectpro.io/project-use-case/cyclegan-implementation-pytorch-for-image-to-image-translation?utm_source=520&utm_medium=fold2)**[View Project](https://www.projectpro.io/project-use-case/cyclegan-implementation-pytorch-for-image-to-image-translation?utm_source=520&utm_medium=fold2)

[View all New Projects](https://www.projectpro.io/projects?utm_source=520&utm_medium=fold2)

#### Q2. How can you create a DataFrame a) using existing RDD, and b) from a CSV file?

Here’s how we can create DataFrame using existing RDDs-

The toDF() function of PySpark RDD is used to construct a DataFrame from an existing RDD. The DataFrame is constructed with the default column names "\_1" and "\_2" to represent the two columns because RDD lacks columns.

dfFromRDD1 = rdd.toDF()

dfFromRDD1.printSchema()

Here, the printSchema() method gives you a database schema without column names-

root

 |-- \_1: string (nullable = true)

 |-- \_2: string (nullable = true)

Use the toDF() function with column names as parameters to pass column names to the DataFrame, as shown below.-

columns = ["language","users\_count"]

dfFromRDD1 = rdd.toDF(columns)

dfFromRDD1.printSchema()

The above code snippet gives you the database schema with the column names-

root

 |-- language: string (nullable = true)

 |-- users: string (nullable = true)

****Upskill yourself for your dream job with industry-level**[big data projects with source code](https://www.projectpro.io/projects/big-data-projects?utm_source=TXTCTA1&utm_medium=ConvertLink&utm_campaign=Blg297" \o "big data projects with source code." \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)**

#### Q3. Explain the use of StructType and StructField classes in PySpark with examples.

The StructType and StructField classes in PySpark are used to define the schema to the DataFrame and create complex columns such as nested struct, array, and map columns. StructType is a collection of StructField objects that determines column name, column data type, field nullability, and metadata.

PySpark imports the StructType class from pyspark.sql.types to describe the DataFrame's structure. The DataFrame's printSchema() function displays StructType columns as "struct."

To define the columns, PySpark offers the pyspark.sql.types import StructField class, which has the column name (String), column type (DataType), nullable column (Boolean), and metadata (MetaData).

Example showing the use of StructType and StructField classes in PySpark-

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType,StructField, StringType, IntegerType

spark = SparkSession.builder.master("local[1]") \

                    .appName('ProjectPro') \

                    .getOrCreate()

data = [("James","","William","36636","M",3000),

    ("Michael","Smith","","40288","M",4000),

    ("Robert","","Dawson","42114","M",4000),

    ("Maria","Jones","39192","F",4000)

  ]

schema = StructType([ \

    StructField("firstname",StringType(),True), \

    StructField("middlename",StringType(),True), \

    StructField("lastname",StringType(),True), \

    StructField("id", StringType(), True), \

    StructField("gender", StringType(), True), \

    StructField("salary", IntegerType(), True) \

  ])

df = spark.createDataFrame(data=data,schema=schema)

df.printSchema()

df.show(truncate=False)

#### Q5. What are the different ways to handle row duplication in a PySpark DataFrame?

There are two ways to handle row duplication in PySpark dataframes. The distinct() function in PySpark is used to drop/remove duplicate rows (all columns) from a DataFrame, while dropDuplicates() is used to drop rows based on one or more columns.

Here’s an example showing how to utilize the distinct() and dropDuplicates() methods-

First, we need to create a sample dataframe.

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.functions import expr

spark = SparkSession.builder.appName('ProjectPro).getOrCreate()

data = [("James", "Sales", 3000), \

    ("Michael", "Sales", 4600), \

    ("Robert", "Sales", 4100), \

    ("Maria", "Finance", 3000), \

    ("James", "Sales", 3000), \

    ("Scott", "Finance", 3300), \

    ("Jen", "Finance", 3900), \

    ("Jeff", "Marketing", 3000), \

    ("Kumar", "Marketing", 2000), \

    ("Saif", "Sales", 4100) \

  ]

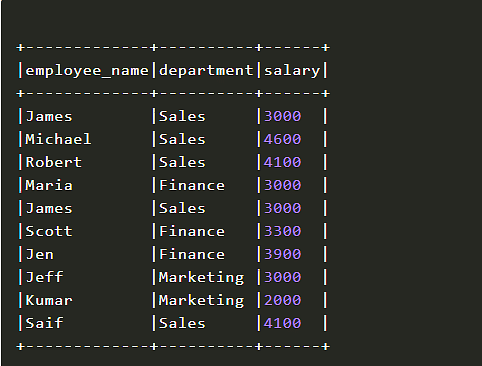
column= ["employee\_name", "department", "salary"]

df = spark.createDataFrame(data = data, schema = column)

df.printSchema()

df.show(truncate=False)

Output-



The record with the employer name Robert contains duplicate rows in the table above. As we can see, there are two rows with duplicate values in all fields and four rows with duplicate values in the department and salary columns.

Below is the entire code for removing duplicate rows-

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.functions import expr

spark = SparkSession.builder.appName('ProjectPro').getOrCreate()

data = [("James", "Sales", 3000), \

    ("Michael", "Sales", 4600), \

    ("Robert", "Sales", 4100), \

    ("Maria", "Finance", 3000), \

    ("James", "Sales", 3000), \

    ("Scott", "Finance", 3300), \

    ("Jen", "Finance", 3900), \

    ("Jeff", "Marketing", 3000), \

    ("Kumar", "Marketing", 2000), \

    ("Saif", "Sales", 4100) \

  ]

column= ["employee\_name", "department", "salary"]

df = spark.createDataFrame(data = data, schema = column)

df.printSchema()

df.show(truncate=False)

#Distinct

distinctDF = df.distinct()

print("Distinct count: "+str(distinctDF.count()))

distinctDF.show(truncate=False)

#Drop duplicates

df2 = df.dropDuplicates()

print("Distinct count: "+str(df2.count()))

df2.show(truncate=False)

#Drop duplicates on selected columns

dropDisDF = df.dropDuplicates(["department","salary"])

print("Distinct count of department salary : "+str(dropDisDF.count()))

dropDisDF.show(truncate=False)

}

****Get FREE Access to**[Data Analytics Example Codes for Data Cleaning, Data Munging, and Data Visualization](https://www.projectpro.io/recipes?utm_source=TXTCTA1&utm_medium=RcpLink&utm_campaign=Blg520" \o "Data Science Example Code Python" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)**

#### Q6. Explain PySpark UDF with the help of an example.

The most important aspect of [Spark SQL](https://www.projectpro.io/article/spark-sql-for-relational-big-data-processing/355" \o "Spark SQL" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) & DataFrame is PySpark UDF (i.e., User Defined Function), which is used to expand PySpark's built-in capabilities. UDFs in PySpark work similarly to UDFs in conventional databases. We write a Python function and wrap it in PySpark SQL udf() or register it as udf and use it on DataFrame and [SQL](https://www.projectpro.io/article/nosql-vs-sql-4-reasons-why-nosql-is-better-for-big-data-applications/86" \o "SQL" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank), respectively, in the case of PySpark.

Example of how we can create a UDF-

First, we need to create a sample dataframe.

spark = SparkSession.builder.appName('ProjectPro').getOrCreate()

column = ["Seqno","Name"]

data = [("1", "john jones"),

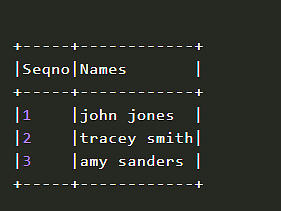
    ("2", "tracey smith"),

    ("3", "amy sanders")]

df = spark.createDataFrame(data=data,schema=column)

df.show(truncate=False)

Output-



The next step is creating a Python function. The code below generates the convertCase() method, which accepts a string parameter and turns every word's initial letter to a capital letter.

def convertCase(str):

    resStr=""

    arr = str.split(" ")

    for x in arr:

       resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "

    return resStr

The final step is converting a Python function to a PySpark UDF.

By passing the function to PySpark SQL udf(), we can convert the convertCase() function to UDF(). The org.apache.spark.sql.functions.udf package contains this function. Before we use this package, we must first import it.

The org.apache.spark.sql.expressions.UserDefinedFunction class object is returned by the PySpark SQL udf() function.

""" Converting function to UDF """

convertUDF = udf(lambda z: convertCase(z),StringType())

#### Q7. Discuss the map() transformation in PySpark DataFrame with the help of an example.

PySpark map or the map() function is an RDD transformation that generates a new RDD by applying 'lambda', which is the transformation function, to each RDD/DataFrame element. RDD map() transformations are used to perform complex operations such as adding a column, changing a column, converting data, and so on. Map transformations always produce the same number of records as the input.

Example of map() transformation in PySpark-

First, we must create an RDD using the list of records.

spark = SparkSession.builder.appName("Map transformation PySpark").getOrCreate()

records = ["Project","Gutenberg’s","Alice’s","Adventures",

"in","Wonderland","Project","Gutenberg’s","Adventures",

"in","Wonderland","Project","Gutenberg’s"]

rdd=spark.sparkContext.parallelize(records)

The map() syntax is-

map(f, preservesPartitioning=False)

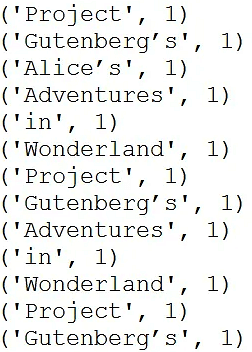
We are adding a new element having value 1 for each element in this PySpark map() example, and the output of the RDD is PairRDDFunctions, which has key-value pairs, where we have a word (String type) as Key and 1 (Int type) as Value.

rdd2=rdd.map(lambda x: (x,1))

for element in rdd2.collect():

    print(element)

Output-



#### Q8. What do you mean by ‘joins’ in PySpark DataFrame? What are the different types of joins?

Joins in PySpark are used to join two DataFrames together, and by linking them together, one may join several DataFrames. INNER Join, LEFT OUTER Join, RIGHT OUTER Join, LEFT ANTI Join, LEFT SEMI Join, CROSS Join, and SELF Join are among the SQL join types it supports.

PySpark Join syntax is-

join(self, other, on=None, how=None)

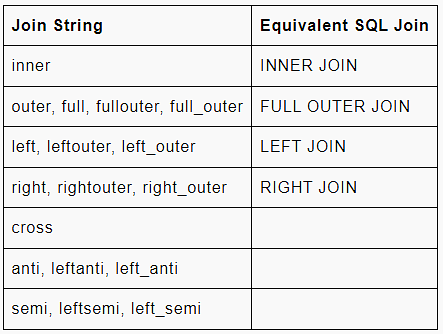
The join() procedure accepts the following parameters and returns a DataFrame-

‘other’: The join's right side;

‘on’: the join column's name;

‘how’: default inner (Options are inner, cross, outer, full, full outer, left, left outer, right, right outer, left semi, and left anti.)

Types of Join in PySpark DataFrame-



#### Q9. What is PySpark ArrayType? Explain with an example.

PySpark ArrayType is a collection data type that extends PySpark's DataType class, which is the superclass for all kinds. The types of items in all ArrayType elements should be the same. The ArraType() method may be used to construct an instance of an ArrayType. It accepts two arguments: valueType and one optional argument valueContainsNull, which specifies whether a value can accept null and is set to True by default. valueType should extend the DataType class in PySpark.

from pyspark.sql.types import StringType, ArrayType

arrayCol = ArrayType(StringType(),False)

#### Q10. What do you understand by PySpark Partition?

Using one or more partition keys, PySpark partitions a large dataset into smaller parts. When we build a DataFrame from a file or table, PySpark creates the DataFrame in memory with a specific number of divisions based on specified criteria. Transformations on partitioned data run quicker since each partition's transformations are executed in parallel. Partitioning in memory (DataFrame) and partitioning on disc (File system) are both supported by PySpark.

****Get More Practice, More**[Big Data and Analytics Projects](https://www.projectpro.io/projects/big-data-projects/apache-spark-projects?utm_source=TXTCTA2&utm_medium=ConvertLink&utm_campaign=Blg520" \o "Big data hadoop project" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)**, and More guidance.Fast-Track Your Career Transition with ProjectPro****

#### Q11. What is meant by PySpark MapType? How can you create a MapType using StructType?

PySpark MapType accepts two mandatory parameters- keyType and valueType, and one optional boolean argument valueContainsNull.

Here’s how to create a MapType with PySpark StructType and StructField. The StructType() accepts a list of StructFields, each of which takes a fieldname and a value type.

from pyspark.sql.types import StructField, StructType, StringType, MapType

schema = StructType([

    StructField('name', StringType(), True),

    StructField('properties', MapType(StringType(),StringType()),True)

])

Now, using the preceding StructType structure, let's construct a DataFrame-

spark= SparkSession.builder.appName('PySpark StructType StructField').getOrCreate()

dataDictionary = [

        ('James',{'hair':'black','eye':'brown'}),

        ('Michael',{'hair':'brown','eye':None}),

        ('Robert',{'hair':'red','eye':'black'}),

        ('Washington',{'hair':'grey','eye':'grey'}),

        ('Jefferson',{'hair':'brown','eye':''})

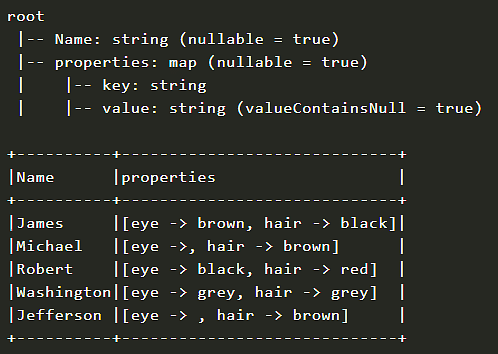
        ]

df = spark.createDataFrame(data=dataDictionary, schema = schema)

df.printSchema()

df.show(truncate=False)

Output-



#### Q12. How can PySpark DataFrame be converted to Pandas DataFrame?

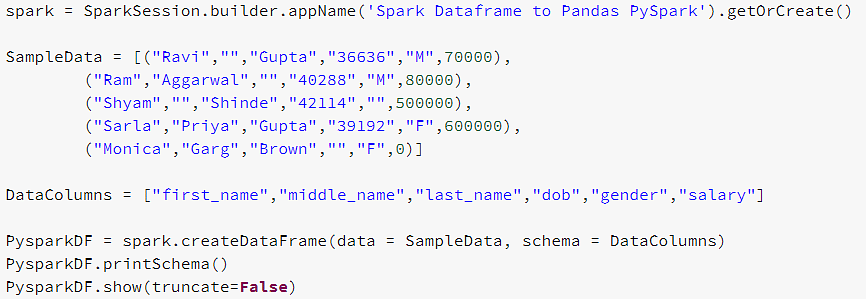
First, you need to learn the difference between the [PySpark](https://www.projectpro.io/apache-spark-tutorial/pyspark-tutorial" \o "PySpark" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) and [Pandas](https://www.projectpro.io/article/python-pandas-dataframe-tutorials/405" \o "Pandas" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank). The key difference between Pandas and PySpark is that PySpark's operations are quicker than Pandas' because of its distributed nature and parallel execution over several cores and computers.

In other words, pandas use a single node to do operations, whereas PySpark uses several computers.

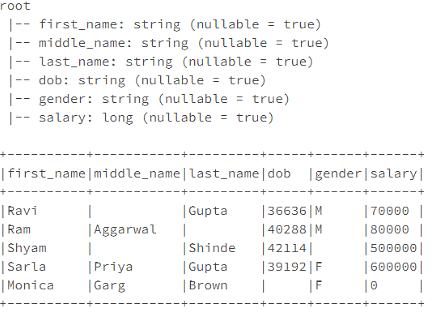
You'll need to transfer the data back to Pandas DataFrame after processing it in PySpark so that you can use it in Machine Learning apps or other Python programs.

Below are the steps to convert PySpark DataFrame into Pandas DataFrame-

You have to start by creating a PySpark DataFrame first.

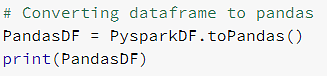


Output-

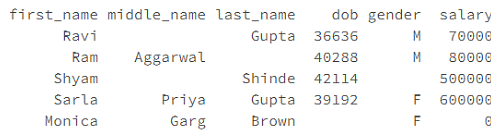


The next step is to convert this PySpark dataframe into Pandas dataframe.

To convert a PySpark DataFrame to a Python Pandas DataFrame, use the toPandas() function. toPandas() gathers all records in a PySpark DataFrame and delivers them to the driver software; it should only be used on a short percentage of the data. When using a bigger dataset, the application fails due to a memory error.



Output-



#### Q13. With the help of an example, show how to employ PySpark ArrayType.

PySpark ArrayType is a data type for collections that extends PySpark's DataType class. The types of items in all ArrayType elements should be the same.

The ArraType() method may be used to construct an instance of an ArrayType. It accepts two arguments: valueType and one optional argument valueContainsNull, which specifies whether a value can accept null and is set to True by default. valueType should extend the DataType class in PySpark.

from pyspark.sql.types import StringType, ArrayType

arrayCol = ArrayType(StringType(),False)

The above example generates a string array that does not allow null values.

#### Q14. What is the function of PySpark's pivot() method?

The pivot() method in PySpark is used to rotate/transpose data from one column into many Dataframe columns and back using the unpivot() function (). Pivot() is an aggregation in which the values of one of the grouping columns are transposed into separate columns containing different data.

To get started, let's make a PySpark DataFrame.

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.functions import expr

#Create spark session

data = [("Banana",1000,"USA"), ("Carrots",1500,"USA"), ("Beans",1600,"USA"), \

      ("Orange",2000,"USA"),("Orange",2000,"USA"),("Banana",400,"China"), \

      ("Carrots",1200,"China"),("Beans",1500,"China"),("Orange",4000,"China"), \

      ("Banana",2000,"Canada"),("Carrots",2000,"Canada"),("Beans",2000,"Mexico")]

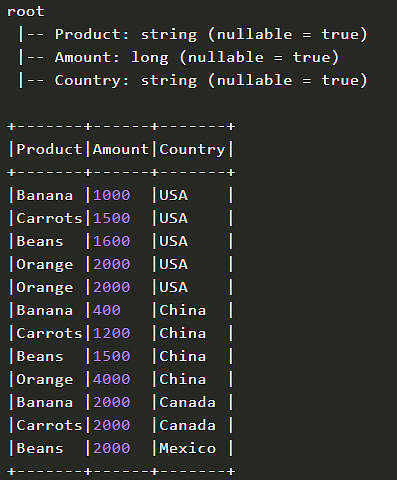
columns= ["Product","Amount","Country"]

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

df.printSchema()

df.show(truncate=False)

Output-



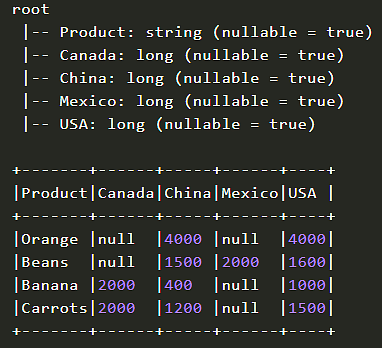
To determine the entire amount of each product's exports to each nation, we'll group by Product, pivot by Country, and sum by Amount.

pivotDF = df.groupBy("Product").pivot("Country").sum("Amount")

pivotDF.printSchema()

pivotDF.show(truncate=False)

This will convert the nations from DataFrame rows to columns, resulting in the output seen below. Wherever data is missing, it is assumed to be null by default.



#### Q15. In PySpark, how do you generate broadcast variables? Give an example.

Broadcast variables in PySpark are read-only shared variables that are stored and accessible on all nodes in a cluster so that processes may access or use them. Instead of sending this information with each job, PySpark uses efficient broadcast algorithms to distribute broadcast variables among workers, lowering communication costs.

The broadcast(v) function of the SparkContext class is used to generate a PySpark Broadcast. This method accepts the broadcast parameter v.

Generating broadcast in PySpark Shell:

broadcastVariable = sc.broadcast(Array(0, 1, 2, 3))

broadcastVariable.value

PySpark RDD Broadcast variable example

spark=SparkSession.builder.appName('SparkByExample.com').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}

broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),

    ("Michael","Rose","USA","NY"),

    ("Robert","Williams","USA","CA"),

    ("Maria","Jones","USA","FL")

  ]

rdd = spark.sparkContext.parallelize(data)

def state\_convert(code):

    return broadcastState.value[code]

res = rdd.map(lambda a: (a[0],a[1],a[2],state\_convert(a{3]))).collect()

print(res)

PySpark DataFrame Broadcast variable example

spark=SparkSession.builder.appName('PySpark broadcast variable').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}

broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),

    ("Michael","Rose","USA","NY"),

    ("Robert","William","USA","CA"),

    ("Maria","Jones","USA","FL")

  ]

columns = ["firstname","lastname","country","state"]

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

df.printSchema()

df.show(truncate=False)

def state\_convert(code):

    return broadcastState.value[code]

res = df.rdd.map(lambda a: (a[0],a[1],a[2],state\_convert(a[3]))).toDF(column)

res.show(truncate=False)

**Explore Categories**

**[Apache Hive Projects](https://www.projectpro.io/projects/big-data-projects/apache-hive-projects?utm_source=520&utm_medium=fold3)** **[Apache Hbase Projects](https://www.projectpro.io/projects/big-data-projects/apache-hbase-projects?utm_source=520&utm_medium=fold3)** **[Apache Pig Projects](https://www.projectpro.io/projects/big-data-projects/apache-pig-projects?utm_source=520&utm_medium=fold3)** **[Hadoop HDFS Projects](https://www.projectpro.io/projects/big-data-projects/hadoop-hdfs-projects?utm_source=520&utm_medium=fold3)** **[Apache Oozie Projects](https://www.projectpro.io/projects/big-data-projects/apache-oozie-projects?utm_source=520&utm_medium=fold3)** **[Apache Impala Projects](https://www.projectpro.io/projects/big-data-projects/apache-impala-projects?utm_source=520&utm_medium=fold3)** **[Apache Flume Projects](https://www.projectpro.io/projects/big-data-projects/apache-flume-projects?utm_source=520&utm_medium=fold3)** **[Spark SQL Projects](https://www.projectpro.io/projects/big-data-projects/spark-sql-projects?utm_source=520&utm_medium=fold3)** **[Spark GraphX Projects](https://www.projectpro.io/projects/big-data-projects/spark-graphx-projects?utm_source=520&utm_medium=fold3)** **[Spark Streaming Projects](https://www.projectpro.io/projects/big-data-projects/spark-streaming-projects?utm_source=520&utm_medium=fold3)** **[Spark MLlib Projects](https://www.projectpro.io/projects/big-data-projects/spark-mllib-projects?utm_source=520&utm_medium=fold3)** **[Apache Zepellin Projects](https://www.projectpro.io/projects/big-data-projects/apache-zepellin-projects?utm_source=520&utm_medium=fold3)** **[Neo4j Projects](https://www.projectpro.io/projects/big-data-projects/neo4j-projects?utm_source=520&utm_medium=fold3)** **[Redis Projects](https://www.projectpro.io/projects/big-data-projects/redis-projects?utm_source=520&utm_medium=fold3)** **[Microsoft Azure Projects](https://www.projectpro.io/projects/big-data-projects/microsoft-azure-projects?utm_source=520&utm_medium=fold3)** **[Google Cloud Projects GCP](https://www.projectpro.io/projects/big-data-projects/google-cloud-projects-gcp?utm_source=520&utm_medium=fold3)** **[AWS Projects](https://www.projectpro.io/projects/big-data-projects/aws-projects?utm_source=520&utm_medium=fold3)**

### ****PySpark Coding Interview Questions****

#### Q1. You have a cluster of ten nodes with each node having 24 CPU cores. The following code works, but it may crash on huge data sets, or at the very least, it may not take advantage of the cluster's full processing capabilities. Which aspect is the most difficult to alter, and how would you go about doing so?

#### def cal(sparkSession: SparkSession): Unit = { val NumNode = 10 val userActivityRdd: RDD[UserActivity] = readUserActivityData(sparkSession) . repartition(NumNode) val result = userActivityRdd .map(e => (e.userId, 1L)) . reduceByKey(\_ + \_) result .take(1000) }

The repartition command creates ten partitions regardless of how many of them were loaded. On large datasets, they might get fairly huge, and they'll almost certainly outgrow the RAM allotted to a single executor.

In addition, each executor can only have one partition. This means that just ten of the 240 executors are engaged (10 nodes with 24 cores, each running one executor).

If the number is set exceptionally high, the scheduler's cost in handling the partition grows, lowering performance. It may even exceed the execution time in some circumstances, especially for extremely tiny partitions.

The optimal number of partitions is between two and three times the number of executors. In the given scenario, 600 = 10 × 24 x 2.5 divisions would be appropriate.

#### Q2. Explain the following code and what output it will yield- case class User(uId: Long, uName: String) case class UserActivity(uId: Long, activityTypeId: Int, timestampEpochSec: Long) val LoginActivityTypeId = 0 val LogoutActivityTypeId = 1 private def readUserData(sparkSession: SparkSession): RDD[User] = { sparkSession.sparkContext.parallelize( Array( User(1, "Doe, John"), User(2, "Doe, Jane"), User(3, "X, Mr.")) ) } private def readUserActivityData(sparkSession: SparkSession): RDD[UserActivity] = { sparkSession.sparkContext.parallelize( Array( UserActivity(1, LoginActivityTypeId, 1514764800L), UserActivity(2, LoginActivityTypeId, 1514808000L), UserActivity(1, LogoutActivityTypeId, 1514829600L), UserActivity(1, LoginActivityTypeId, 1514894400L)) ) } def calculate(sparkSession: SparkSession): Unit = { val userRdd: RDD[(Long, User)] = readUserData(sparkSession).map(e => (e.userId, e)) val userActivityRdd: RDD[(Long, UserActivity)] = readUserActivityData(sparkSession).map(e => (e.userId, e)) val result = userRdd .leftOuterJoin(userActivityRdd) .filter(e => e.\_2.\_2.isDefined && e.\_2.\_2.get.activityTypeId == LoginActivityTypeId) .map(e => (e.\_2.\_1.uName, e.\_2.\_2.get.timestampEpochSec)) .reduceByKey((a, b) => if (a < b) a else b) result .foreach(e => println(s"${e.\_1}: ${e.\_2}")) }

The primary function, calculate, reads two pieces of data. (They are given in this case from a constant inline data structure that is transformed to a distributed dataset using parallelize.) Each of them is transformed into a tuple by the map, which consists of a userId and the item itself. To combine the two datasets, the userId is utilised.

All users' login actions are filtered out of the combined dataset. The uName and the event timestamp are then combined to make a tuple.

This is eventually reduced down to merely the initial login record per user, which is then sent to the console.

The following will be the yielded output-

Doe, John: 1514764800

Doe, Jane: 1514808000

#### Q3. The code below generates two dataframes with the following structure: DF1: uId, uName DF2: uId, pageId, timestamp, eventType. Join the two dataframes using code and count the number of events per uName. It should only output for users who have events in the format uName; totalEventCount.

#### def calculate(sparkSession: SparkSession): Unit = { val UIdColName = "uId" val UNameColName = "uName" val CountColName = "totalEventCount" val userRdd: DataFrame = readUserData(sparkSession) val userActivityRdd: DataFrame = readUserActivityData(sparkSession) val res = userRdd .repartition(col(UIdColName)) // ??????????????? . select(col(UNameColName))// ??????????????? result.show() }

This is how the code looks:

def calculate(sparkSession: SparkSession): Unit = {

 val UIdColName = "uId"

 val UNameColName = "uName"

 val CountColName = "totalEventCount"

 val userRdd: DataFrame = readUserData(sparkSession)

 val userActivityRdd: DataFrame = readUserActivityData(sparkSession)

 val result = userRdd

   .repartition(col(UIdColName))

   .join(userActivityRdd, UIdColName)

   .select(col(UNameColName))

   .groupBy(UNameColName)

   .count()

   .withColumnRenamed("count", CountColName)

 result.show()

}

#### Q4. Please indicate which parts of the following code will run on the master and which parts will run on each worker node.

#### val formatter: DateTimeFormatter = DateTimeFormatter.ofPattern("yyyy/MM") def getEventCountOnWeekdaysPerMonth(data: RDD[(LocalDateTime, Long)]): Array[(String, Long)] = { val res = data .filter(e => e.\_1.getDayOfWeek.getValue < DayOfWeek.SATURDAY.getValue) . map(mapDateTime2Date) . reduceByKey(\_ + \_) . collect() result . map(e => (e.\_1.format(formatter), e.\_2)) } private def mapDateTime2Date(v: (LocalDateTime, Long)): (LocalDate, Long) = { (v.\_1.toLocalDate.withDayOfMonth(1), v.\_2) }

The driver application is responsible for calling this function. The DAG is defined by the assignment to the result value, as well as its execution, which is initiated by the collect() operation. The worker nodes handle all of this (including the logic of the method mapDateTime2Date). Because the result value that is gathered on the master is an array, the map performed on this value is also performed on the master.

#### Q5. What are the elements used by the GraphX library, and how are they generated from an RDD? To determine page rankings, fill in the following code-

#### def calculate(sparkSession: SparkSession): Unit = { val pageRdd: RDD[(???, Page)] = readPageData(sparkSession) . map(e => (e.pageId, e)) . cache() val pageReferenceRdd: RDD[???[PageReference]] = readPageReferenceData(sparkSession) val graph = Graph(pageRdd, pageReferenceRdd) val PageRankTolerance = 0.005 val ranks = graph.??? ranks.take(1000).foreach(print) } The output yielded will be a list of tuples: (1,1.4537951595091907) (2,0.7731024202454048) (3,0.7731024202454048)

Vertex, and Edge objects are supplied to the Graph object as RDDs of type RDD[VertexId, VT] and RDD[Edge[ET]] respectively (where VT and ET are any user-defined types associated with a given Vertex or Edge). For Edge type, the constructor is Edge[ET](srcId: VertexId, dstId: VertexId, attr: ET). VertexId is just an alias for Long.

**Get confident to build end-to-end projects.**

Access to a curated library of 250+ end-to-end industry projects with solution code, videos and tech support.

**[Request a demo](https://www.projectpro.io/project/project-demo?utm_source=520&utm_medium=fold4" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)**

### ****PySpark Interview Questions for Data Engineer****

#### Q1. Under what scenarios are Client and Cluster modes used for deployment?

Cluster mode should be utilized for deployment if the client computers are not near the cluster. This is done to prevent the network delay that would occur in Client mode while communicating between executors. In case of Client mode, if the machine goes offline, the entire operation is lost.

Client mode can be utilized for deployment if the client computer is located within the cluster. There will be no network latency concerns because the computer is part of the cluster, and the cluster's maintenance is already taken care of, so there is no need to be concerned in the event of a failure.

#### Q2.How is Apache Spark different from MapReduce?

|  |  |
| --- | --- |
| MapReduce | Apache Spark |
| Only batch-wise data processing is done using MapReduce. | Apache Spark can handle data in both real-time and batch mode. |
| The data is stored in HDFS (Hadoop Distributed File System), which takes a long time to retrieve. | Spark saves data in memory (RAM), making data retrieval quicker and faster when needed. |
| MapReduce is a high-latency framework since it is heavily reliant on disc. | Spark is a low-latency computation platform because it offers in-memory data storage and caching. |

#### Q2. Write a spark program to check whether a given keyword exists in a huge text file or not?

def keywordExists(line):

   if (line.find(“my\_keyword”) > -1):

       return 1

   return 0

lines = sparkContext.textFile(“sample\_file.txt”);

isExist = lines.map(keywordExists);

sum=isExist.reduce(sum);

print(“Found” if sum>0 else “Not Found”)

#### Q3. What is meant by Executor Memory in PySpark?

Spark executors have the same fixed core count and heap size as the applications created in Spark. The heap size relates to the memory used by the Spark executor, which is controlled by the -executor-memory flag's property spark.executor.memory. On each worker node where Spark operates, one executor is assigned to it. The executor memory is a measurement of the memory utilized by the application's worker node.

#### Q4. List some of the functions of SparkCore.

The core engine for large-scale distributed and parallel data processing is SparkCore. The distributed execution engine in the Spark core provides APIs in Java, Python, and [Scala](https://www.projectpro.io/article/scala-interview-questions-and-answers-for-spark-developers/302" \o "Scala" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) for constructing distributed ETL applications.

Memory management, task monitoring, fault tolerance, storage system interactions, work scheduling, and support for all fundamental I/O activities are all performed by Spark Core. Additional libraries on top of Spark Core enable a variety of SQL, streaming, and machine learning applications.

They are in charge of:

Fault Recovery

Interactions between memory management and storage systems

Monitoring, scheduling, and distributing jobs

Fundamental I/O functions

#### Q5. What are some of the drawbacks of incorporating Spark into applications?

Despite the fact that Spark is a strong data processing engine, there are certain drawbacks to utilizing it in applications.

When compared to MapReduce or Hadoop, Spark consumes greater storage space, which may cause memory-related issues.

Spark can be a constraint for cost-effective large data processing since it uses "in-memory" calculations.

When working in cluster mode, files on the path of the local filesystem must be available at the same place on all worker nodes, as the task execution shuffles across different worker nodes based on resource availability. All worker nodes must copy the files, or a separate network-mounted file-sharing system must be installed.

****Build an Awesome Job Winning Project Portfolio with Solved**[End-to-End Big Data Projects](https://www.projectpro.io/projects/big-data-projects/apache-spark-projects?utm_source=TXTCTA3&utm_medium=ConvertLink&utm_campaign=Blg520" \o "Big data spark projects" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)**

#### Q6. How can data transfers be kept to a minimum while using PySpark?

The process of shuffling corresponds to data transfers. Spark applications run quicker and more reliably when these transfers are minimized. There are quite a number of approaches that may be used to reduce them. They are as follows:

Using broadcast variables improves the efficiency of joining big and small RDDs.

Accumulators are used to update variable values in a parallel manner during execution.

Another popular method is to prevent operations that cause these reshuffles.

#### Q7. What are Sparse Vectors? What distinguishes them from dense vectors?

Sparse vectors are made up of two parallel arrays, one for indexing and the other for storing values. These vectors are used to save space by storing non-zero values. E.g.- val sparseVec: Vector = Vectors.sparse(5, Array(0, 4), Array(1.0, 2.0))

The vector in the above example is of size 5, but the non-zero values are only found at indices 0 and 4.

When there are just a few non-zero values, sparse vectors come in handy. If there are just a few zero values, dense vectors should be used instead of sparse vectors, as sparse vectors would create indexing overhead, which might affect performance.

The following is an example of a dense vector:

val denseVec = Vectors.dense(4405d,260100d,400d,5.0,4.0,198.0,9070d,1.0,1.0,2.0,0.0)

The usage of sparse or dense vectors has no effect on the outcomes of calculations, but when they are used incorrectly, they have an influence on the amount of memory needed and the calculation time.

#### Q8. What role does Caching play in Spark Streaming?

The partition of a data stream's contents into batches of X seconds, known as DStreams, is the basis of [Spark Streaming](https://www.projectpro.io/article/taming-big-data-with-spark-streaming-for-real-time-data-processing/337" \o "Spark Streaming" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank). These DStreams allow developers to cache data in memory, which may be particularly handy if the data from a DStream is utilized several times. The cache() function or the persist() method with proper persistence settings can be used to cache data. For input streams receiving data through networks such as Kafka, Flume, and others, the default persistence level setting is configured to achieve data replication on two nodes to achieve fault tolerance.

Cache method-

val cacheDf = dframe.cache()

Persist method-

val persistDf = dframe.persist(StorageLevel.MEMORY\_ONLY)

The following are the key benefits of caching:

Cost-effectiveness: Because Spark calculations are costly, caching aids in data reuse, which leads to reuse computations, lowering the cost of operations.

Time-saving: By reusing computations, we may save a lot of time.

More Jobs Achieved: Worker nodes may perform/execute more jobs by reducing computation execution time.

#### Q9. What API does PySpark utilize to implement graphs?

Spark RDD is extended with a robust API called GraphX, which supports graphs and graph-based calculations. The Resilient Distributed Property Graph is an enhanced property of Spark RDD that is a directed multi-graph with many parallel edges. User-defined characteristics are associated with each edge and vertex. Multiple connections between the same set of vertices are shown by the existence of parallel edges. GraphX offers a collection of operators that can allow graph computing, such as subgraph, mapReduceTriplets, joinVertices, and so on. It also offers a wide number of graph builders and algorithms for making graph analytics chores easier.

#### Q10. What is meant by Piping in PySpark?

According to the UNIX Standard Streams, Apache Spark supports the pipe() function on RDDs, which allows you to assemble distinct portions of jobs that can use any language. The RDD transformation may be created using the pipe() function, and it can be used to read each element of the RDD as a String. These may be altered as needed, and the results can be presented as Strings.

#### Q11. What are the various levels of persistence that exist in PySpark?

Spark automatically saves intermediate data from various shuffle processes. However, it is advised to use the RDD's persist() function. There are many levels of persistence for storing RDDs on memory, disc, or both, with varying levels of replication. The following are the persistence levels available in Spark:

MEMORY ONLY: This is the default persistence level, and it's used to save RDDs on the JVM as deserialized Java objects. In the event that the RDDs are too large to fit in memory, the partitions are not cached and must be recomputed as needed.

MEMORY AND DISK: On the JVM, the RDDs are saved as deserialized Java objects. In the event that memory is inadequate, partitions that do not fit in memory will be kept on disc, and data will be retrieved from the drive as needed.

MEMORY ONLY SER: The RDD is stored as One Byte per partition serialized Java Objects.

DISK ONLY: RDD partitions are only saved on disc.

OFF HEAP: This level is similar to MEMORY ONLY SER, except that the data is saved in off-heap memory.

The persist() function has the following syntax for employing persistence levels:

df.persist(StorageLevel.)

#### Q12. What steps are involved in calculating the executor memory?

Suppose you have the following details regarding the cluster:

No. of nodes = 10

No. of cores in each node = 15 cores

RAM of each node = 61GB

We use the following method to determine the number of cores:

No. of cores = How many concurrent tasks the executor can handle.

As a rule of thumb, 5 is the best value.

Hence, we use the following method to determine the number of executors:

No. of executors = No. of cores/Concurrent Task

                   = 15/5

                   = 3

No. of executors = No. of nodes \* No. of executors in each node

                   = 10 \* 3

                   = 30 executors per Spark job

#### Q13. Do we have a checkpoint feature in Apache Spark?

Yes, there is an API for checkpoints in Spark. The practice of checkpointing makes streaming apps more immune to errors. We can store the data and metadata in a checkpointing directory. If there’s a failure, the spark may retrieve this data and resume where it left off.

In Spark, checkpointing may be used for the following data categories-

Metadata checkpointing: Metadata rmeans information about information. It refers to storing metadata in a fault-tolerant storage system such as HDFS. You can consider configurations, DStream actions, and unfinished batches as types of metadata.

Data checkpointing: Because some of the stateful operations demand it, we save the RDD to secure storage. The RDD for the next batch is defined by the RDDs from previous batches in this case.

#### Q14. In Spark, how would you calculate the total number of unique words?

1. Open the text file in RDD mode:

sc.textFile(“[hdfs://Hadoop/user/sample\_file.txt](hdfs://Hadoop/user/sample_file.txt" \o "hdfs://Hadoop/user/sample_file.txt" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)”);

2. A function that converts each line into words:

def toWords(line):

return line.split();

3. As a flatMap transformation, run the toWords function on each item of the RDD in Spark:

words = line.flatMap(toWords);

4. Create a (key,value) pair for each word:

def toTuple(word):

return (word, 1);

wordTuple = words.map(toTuple);

5. Run the reduceByKey() command:

def sum(x, y):

return x+y:

counts = wordsTuple.reduceByKey(sum)

6. Print:

counts.collect()

#### Q15. List some of the benefits of using PySpark.

PySpark is a specialized in-memory distributed processing engine that enables you to handle data in a distributed fashion effectively.

PySpark-based programs are 100 times quicker than traditional apps.

You can learn a lot by utilizing PySpark for data intake processes. PySpark can handle data from Hadoop HDFS, Amazon S3, and a variety of other file systems.

Through the use of Streaming and Kafka, PySpark is also utilized to process real-time data.

You can use PySpark streaming to swap data between the file system and the socket.

PySpark contains machine learning and graph libraries by chance.

### ****PySpark Data Science Interview Questions****

#### Q1. What distinguishes [Apache Spark](https://www.projectpro.io/article/top-50-spark-interview-questions-and-answers-for-2021/208" \l "toc-4" \o "Apache Spark" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) from other programming languages?

High Data Processing Speed: By decreasing read-write operations to disc, Apache Spark aids in achieving a very high data processing speed. When doing in-memory computations, the speed is about 100 times quicker, and when performing disc computations, the speed is 10 times faster.

Dynamic in nature: Spark's dynamic nature comes from 80 high-level operators, making developing parallel applications a breeze.

In-memory Computing Ability: Spark's in-memory computing capability, which is enabled by its DAG execution engine, boosts data processing speed. This also allows for data caching, which reduces the time it takes to retrieve data from the disc.

Fault Tolerance: RDD is used by Spark to support fault tolerance. Spark RDDs are abstractions that are meant to accommodate worker node failures while ensuring that no data is lost.

Stream Processing: Spark offers real-time stream processing. The difficulty with the previous MapReduce architecture was that it could only handle data that had already been created.

#### Q2. Explain RDDs in detail.

Resilient Distribution Datasets (RDD) are a collection of fault-tolerant functional units that may run simultaneously. RDDs are data fragments that are maintained in memory and spread across several nodes. In an RDD, all partitioned data is distributed and consistent.

There are two types of RDDs available:

Hadoop datasets- Those datasets that apply a function to each file record in the Hadoop Distributed File System (HDFS) or another file storage system.

Parallelized Collections- Existing RDDs that operate in parallel with each other.

#### Q3. Mention some of the major advantages and disadvantages of PySpark.

Some of the major advantages of using PySpark are-

Writing parallelized code is effortless.

Keeps track of synchronization points and errors.

Has a lot of useful built-in algorithms.

Some of the disadvantages of using PySpark are-

Managing an issue with MapReduce may be difficult at times.

It is inefficient when compared to alternative programming paradigms.

#### Q4. Explain the profilers which we use in PySpark.

PySpark allows you to create custom profiles that may be used to build predictive models. In general, profilers are calculated using the minimum and maximum values of each column. It is utilized as a valuable data review tool to ensure that the data is accurate and appropriate for future usage.

The following methods should be defined or inherited for a custom profiler-

profile- this is identical to the system profile.

add- this is a command that allows us to add a profile to an existing accumulated profile.

dump- saves all of the profiles to a path.

stats- returns the stats that have been gathered.

#### Q5. List some recommended practices for making your PySpark data science workflows better.

Avoid dictionaries: If you use Python data types like dictionaries, your code might not be able to run in a distributed manner. Consider adding another column to a dataframe that may be used as a filter instead of utilizing keys to index entries in a dictionary. This proposal also applies to Python types that aren't distributable in PySpark, such as lists.

Limit the use of Pandas: using toPandas causes all data to be loaded into memory on the driver node, preventing operations from being run in a distributed manner. When data has previously been aggregated, and you wish to utilize conventional Python plotting tools, this method is appropriate, but it should not be used for larger dataframes.

Minimize eager operations: It's best to avoid eager operations that draw whole dataframes into memory if you want your pipeline to be as scalable as possible. Reading in CSVs, for example, is an eager activity, thus I stage the dataframe to S3 as Parquet before utilizing it in further pipeline steps.

### ****Advanced PySpark Interview Questions and Answers****

#### Q1. Discuss PySpark SQL in detail.

PySpark SQL is a structured data library for Spark. PySpark SQL, in contrast to the PySpark RDD API, offers additional detail about the data structure and operations. It comes with a programming paradigm- ‘DataFrame.’

A DataFrame is an immutable distributed columnar data collection. DataFrames can process huge amounts of organized data (such as relational databases) and semi-structured data (JavaScript Object Notation or JSON).

After creating a dataframe, you can interact with data using SQL syntax/queries.

The first step in using PySpark SQL is to use the createOrReplaceTempView() function to create a temporary table on DataFrame. The table is available throughout SparkSession via the sql() method. You can delete the temporary table by ending the SparkSession.

Example of PySpark SQL-

import findspark

findspark.init()

import pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

df = spark.sql('''select 'spark' as hello ''')

df.show()

#### Q2. Explain the different persistence levels in PySpark.

Persisting (or caching) a dataset in memory is one of PySpark's most essential features. The different levels of persistence in PySpark are as follows-

|  |  |
| --- | --- |
| Level | Purpose |
| MEMORY\_ONLY | This level stores deserialized Java objects in the JVM. It is the default persistence level in PySpark. |
| MEMORY\_AND\_DISK | This level stores RDD as deserialized Java objects. If the RDD is too large to reside in memory, it saves the partitions that don't fit on the disk and reads them as needed. |
| MEMORY\_ONLY\_SER | It stores RDD in the form of serialized Java objects. Although this level saves more space in the case of fast serializers, it demands more CPU capacity to read the RDD. |
| MEMORY\_AND\_DISK\_SER | This level acts similar to MEMORY ONLY SER, except instead of recomputing partitions on the fly each time they're needed, it stores them on disk. |
| DISK\_ONLY | It only saves RDD partitions on the disk. |
| MEMORY\_ONLY\_2, MEMORY\_AND\_DISK\_2, etc. | These levels function the same as others. They copy each partition on two cluster nodes. |
| OFF\_HEAP | This level requires off-heap memory to store RDD. |

**Most Watched Projects**

**[Build an Analytical Platform for eCommerce using AWS Services](https://www.projectpro.io/project-use-case/analytical-platform-for-ecommerce-using-aws?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/analytical-platform-for-ecommerce-using-aws?utm_source=520&utm_medium=fold5)

**[Learn to Build a Polynomial Regression Model from Scratch](https://www.projectpro.io/project-use-case/polynomial-regression-model-in-python-from-scratch?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/polynomial-regression-model-in-python-from-scratch?utm_source=520&utm_medium=fold5)

**[Hands-On Real Time PySpark Project for Beginners](https://www.projectpro.io/project-use-case/real-time-end-to-end-pyspark-project-for-beginners?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/real-time-end-to-end-pyspark-project-for-beginners?utm_source=520&utm_medium=fold5)

**[Customer Churn Prediction Analysis using Ensemble Techniques](https://www.projectpro.io/project-use-case/customer-churn-prediction-using-python?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/customer-churn-prediction-using-python?utm_source=520&utm_medium=fold5)

**[Build an AWS ETL Data Pipeline in Python on YouTube Data](https://www.projectpro.io/project-use-case/aws-etl-data-pipeline-python-example?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/aws-etl-data-pipeline-python-example?utm_source=520&utm_medium=fold5)

**[Build an Analytical Platform for eCommerce using AWS Services](https://www.projectpro.io/project-use-case/analytical-platform-for-ecommerce-using-aws?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/analytical-platform-for-ecommerce-using-aws?utm_source=520&utm_medium=fold5)

**[Learn to Build a Polynomial Regression Model from Scratch](https://www.projectpro.io/project-use-case/polynomial-regression-model-in-python-from-scratch?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/polynomial-regression-model-in-python-from-scratch?utm_source=520&utm_medium=fold5)

**[Hands-On Real Time PySpark Project for Beginners](https://www.projectpro.io/project-use-case/real-time-end-to-end-pyspark-project-for-beginners?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/real-time-end-to-end-pyspark-project-for-beginners?utm_source=520&utm_medium=fold5)

**[Customer Churn Prediction Analysis using Ensemble Techniques](https://www.projectpro.io/project-use-case/customer-churn-prediction-using-python?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/customer-churn-prediction-using-python?utm_source=520&utm_medium=fold5)

**[Build an AWS ETL Data Pipeline in Python on YouTube Data](https://www.projectpro.io/project-use-case/aws-etl-data-pipeline-python-example?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/aws-etl-data-pipeline-python-example?utm_source=520&utm_medium=fold5)

**[Build an Analytical Platform for eCommerce using AWS Services](https://www.projectpro.io/project-use-case/analytical-platform-for-ecommerce-using-aws?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/analytical-platform-for-ecommerce-using-aws?utm_source=520&utm_medium=fold5)

**[Learn to Build a Polynomial Regression Model from Scratch](https://www.projectpro.io/project-use-case/polynomial-regression-model-in-python-from-scratch?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/polynomial-regression-model-in-python-from-scratch?utm_source=520&utm_medium=fold5)

**[Hands-On Real Time PySpark Project for Beginners](https://www.projectpro.io/project-use-case/real-time-end-to-end-pyspark-project-for-beginners?utm_source=520&utm_medium=fold5)**[View Project](https://www.projectpro.io/project-use-case/real-time-end-to-end-pyspark-project-for-beginners?utm_source=520&utm_medium=fold5)

[View all Most Watched Projects](https://www.projectpro.io/projects?utm_source=520&utm_medium=fold5)

#### Q3. What do you mean by checkpointing in PySpark?

A streaming application must be available 24 hours a day, seven days a week, and must be resistant to errors external to the application code (e.g., system failures, JVM crashes, etc.).

The process of checkpointing makes streaming applications more tolerant of failures. You can save the data and metadata to a checkpointing directory.

Checkpointing can be of two types- Metadata checkpointing and Data checkpointing.

Metadata checkpointing allows you to save the information that defines the streaming computation to a fault-tolerant storage system like HDFS. This helps to recover data from the failure of the streaming application's driver node.

Data checkpointing entails saving the created RDDs to a secure location. Several stateful computations combining data from different batches require this type of checkpoint.

#### Q4. In PySpark, how would you determine the total number of unique words?

Open the text file in RDD mode:

sc.textFile(“[hdfs://Hadoop/user/test\_file.txt](hdfs://Hadoop/user/test_file.txt" \o "hdfs://Hadoop/user/test_file.txt" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)”);

Write a function that converts each line into a single word:

def toWords(line):

return line.split();

Run the toWords function on each member of the RDD in Spark:  
words = line.flatMap(toWords);

Generate a (key, value) for each word:

def toTuple(word):

return (word, 1);

wordTuple = words.map(toTuple);

Run the reduceByKey() command:

def sum(x, y):

return x+y:

counts = wordsTuple.reduceByKey(sum)

Print it out:

counts.collect()

#### Q5. Explain PySpark Streaming. How do you use the TCP/IP Protocol to stream data

Spark Streaming is a feature of the core Spark API that allows for scalable, high-throughput, and fault-tolerant live data stream processing.

It entails data ingestion from various sources, including Kafka, Kinesis, TCP connections, and data processing with complicated algorithms using high-level functions like map, reduce, join, and window.



Furthermore, it can write data to filesystems, databases, and live dashboards.

We can use the readStream.format("socket") method of the Spark session object for reading data from a TCP socket and specifying the streaming source host and port as parameters, as illustrated in the code below:

from pyspark import SparkContext

from pyspark.streaming import StreamingContext

sc = SparkContext("local[2]", "NetworkWordCount")

ssc = StreamingContext(sc, 1)

lines = ssc.socketTextStream("localhost", 9999)

#### Q6.What do you understand by Lineage Graph in PySpark?

The Spark lineage graph is a collection of RDD dependencies. There are separate lineage graphs for each Spark application.

The lineage graph recompiles RDDs on-demand and restores lost data from persisted RDDs.

An RDD lineage graph helps you to construct a new RDD or restore data from a lost persisted RDD.

It's created by applying modifications to the RDD and generating a consistent execution plan.

#### Q7. Outline some of the features of PySpark SQL.

User-Defined Functions- To extend the Spark functions, you can define your own column-based transformations.

Standard JDBC/ODBC Connectivity- Spark SQL libraries allow you to connect to Spark SQL using regular JDBC/ODBC connections and run queries (table operations) on structured data.

Data Transformations- For transformations, Spark's RDD API offers the highest quality performance. Spark takes advantage of this functionality by converting SQL queries to RDDs for transformations.

Performance- Due to its in-memory processing, Spark SQL outperforms Hadoop by allowing for more iterations over datasets.

Relational Processing- Spark brought relational processing capabilities to its functional programming capabilities with the advent of SQL.

#### Q8. Define the role of Catalyst Optimizer in PySpark.

Apache Spark relies heavily on the Catalyst optimizer. It improves structural queries expressed in SQL or via the DataFrame/Dataset APIs, reducing program runtime and cutting costs.

The Spark Catalyst optimizer supports both rule-based and cost-based optimization.

Rule-based optimization involves a set of rules to define how to execute the query.

Cost-based optimization involves developing several plans using rules and then calculating their costs.

Catalyst optimizer also handles various Big data challenges like semistructured data and advanced analytics.

#### Q9. Mention the various operators in PySpark GraphX.

Property Operators- These operators create a new graph with the user-defined map function modifying the vertex or edge characteristics. In these operators, the graph structure is unaltered. This is a significant feature of these operators since it allows the generated graph to maintain the original graph's structural indices.

Structural Operators- GraphX currently only supports a few widely used structural operators. The reverse operator creates a new graph with reversed edge directions. The subgraph operator returns a graph with just the vertices and edges that meet the vertex predicate. The mask operator creates a subgraph by returning a graph with all of the vertices and edges found in the input graph. The groupEdges operator merges parallel edges.

Join Operators- The join operators allow you to join data from external collections (RDDs) to existing graphs. For example, you might want to combine new user attributes with an existing graph or pull vertex properties from one graph into another.

#### Q10. Consider the following scenario: you have a large text file. How will you use PySpark to see if a specific keyword exists?

lines = sc.textFile(“[hdfs://Hadoop/user/test\_file.txt](hdfs://Hadoop/user/test_file.txt" \o "hdfs://Hadoop/user/test_file.txt" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank)”);

def isFound(line):

if line.find(“my\_keyword”) > -1

return 1

return 0

foundBits = lines.map(isFound);

sum = foundBits.reduce(sum);

if sum > 0:

print “Found”

else:

print “Not Found”;

### ****PySpark Practice Problems | Scenario Based Interview Questions and Answers****

#### Q1. The given file has a delimiter ~|. How will you load it as a spark DataFrame?

Important: Instead of using sparkContext(sc), use sparkSession (spark).

Name ~|Age

Azarudeen, Shahul~|25

Michel, Clarke ~|26

Virat, Kohli ~|28

Andrew, Simond ~|37

George, Bush~|59

Flintoff, David ~|12

Answer- import findspark

findspark.init()

from pyspark.sql import Sparksession, types

spark = Sparksession.builder.master("local").appliame("scenario based")\

-getorcreate()

sc=spark.sparkContext

dfaspark.read.text("input.csv")

df.show(truncate=0)

header=df.first()[0]

schema=header.split(-')

df\_imput=df.filter(df['value'] l= header).rdd.map(lambda x: x[0]. split('-|')).toDF (schema)

df\_input.show(truncate=0)

#### Q2. How will you merge two files – File1 and File2 – into a single DataFrame if they have different schemas?

File -1:

Name|Age

Azarudeen, Shahul|25

Michel, Clarke|26

Virat, Kohli|28

Andrew, Simond|37

File -2:

Name|Age|Gender

Rabindra, Tagore |32|Male

Madona, Laure | 59|Female

Flintoff, David|12|Male

Ammie, James| 20|Female

Answer- import findspark

findspark.init()

from pyspark.sql import SparkSession, types

spark = SparkSession.builder.master("local").appName('Modes of Dataframereader')\

.getorCreate()

sc=spark.sparkContext

df1=spark.read.option("delimiter","|").csv('input.csv')

df2=spark.read.option("delimiter","|").csv("input2.csv",header=True)

from pyspark.sql.functions import lit

df\_add=df1.withColumn("Gender",lit("null"))

df\_add. union(df2).show()

For the Union-

from pyspark.sql.types import \*

schema=StructType(

    [

        StructField("Name",StringType(), True),

        StructField("Age",StringType(), True),

        StructField("Gender",StringType(),True),

    ]

)

df3=spark.read.option("delimiter","|").csv("input.csv",header=True, schema=schema)

df4=spark.read.option("delimiter","|").csv("input2.csv", header=True, schema=schema)

df3.union(df4).show()

#### Q3. Examine the following file, which contains some corrupt/bad data. What will you do with such data, and how will you import them into a Spark Dataframe?

Emp\_no, Emp\_name, Department

101, Murugan, HealthCare

Invalid Entry, Description: Bad Record entry

102, Kannan, Finance

103, Mani, IT

Connection lost, Description: Poor Connection

104, Pavan, HR

Bad Record, Description: Corrupt record

Answer-

import findspark

findspark.init()

from pyspark. sql import Sparksession, types

spark = Sparksession.builder.master("local").appName( "Modes of Dataframereader')\

.getorcreate()

sc=spark. sparkContext

from pyspark.sql.types import \*

schm structiype([

structField("col\_1",stringType(), True),

StructField("col\_2",stringType(), True),

structrield("col",stringtype(), True),

])

df=spark.read.option("mode", "DROPMALFORMED").csv('input1.csv', header=True, schema=schm)

df. show()

#### Q4. Consider a file containing an Education column that includes an array of elements, as shown below. Using Spark Dataframe, convert each element in the array to a record.

Name| Age | EducaÈ›ion

Azar|25| MBA,BE,HSC

Hari|32|

Kumar|35|ME,BE,Diploma

Answer-

import findspark

findspark.init()

from pyspark.sql import SparkSession, types

spark = SparkSession.builder.master("local").appName('scenario based')\

.getorCreate()

sc=spark.sparkContext

in\_df=spark.read.option("delimiter","|").csv("input4.csv", header-True)

in\_df.show()

from pyspark.sql.functions import posexplode\_outer, split

in\_df.withColumn("Qualification", explode\_outer(split("Education",","))).show()

in\_df.select("\*", posexplode\_outer(split("Education",","))).withColumnRenamed ("col", "Qualification").withColumnRenamed ("pos", "Index").drop(“Education”).show()

****Q5. Give the output for the following-****

input.csv:

101,Azar,finance

102,Mani,HR

103,Raj,IT

in\_rdd=sc.textFile('input.csv')

map\_rdd=in\_rdd.map(lambda x: x.split(','))

map\_rdd.count()

in\_rdd=sc.textFile('input.csv')

map\_rdd=in\_rdd.flatMap(lambda x: x.split(','))

map\_rdd.count()

Answer-

import findspark

findspark.init()

from pyspark.sql import SparkSession

spark=SparkSession.builder.master("local").appName( "map").getOrCreate()

sc=spark.sparkContext

in\_rdd=sc.textFile('inputfile.txt')

map\_rdd=in\_rdd.map(lambda x: x.split(','))

map\_rdd.collect()

map\_rdd.count ()

For the above code, the output is 3.

import findspark

findspark.init()

from pyspark.sql import SparkSession

spark=SparkSession.builder.master("local").appName( "map").getOrCreate()

sc=spark.sparkContext

in\_rdd=sc.textFile('inputfile.txt')

flat\_map\_rdd=in\_rdd.flatMap(lambda x: x.split(','))

for i in flat\_map\_rdd.collect():

print(i)

in\_rdd.collect()

flat\_map\_rdd.count()

For the above code, the output is 9.

### ****Capgemini PySpark Interview Questions****

#### Q1. What are SparkFiles in Pyspark?

PySpark provides the reliability needed to upload our files to Apache Spark. This is accomplished by using sc.addFile, where 'sc' stands for SparkContext. We use SparkFiles.net to acquire the directory path.

We use the following methods in SparkFiles to resolve the path to the files added using SparkContext.addFile():

get(filename),

getrootdirectory()

#### Q2. What is SparkConf in PySpark? List a few attributes of SparkConf.

SparkConf aids in the setup and settings needed to execute a spark application locally or in a cluster. To put it another way, it offers settings for running a Spark application. The following are some of SparkConf's most important features:

set(key, value): This attribute aids in the configuration property setting.

setSparkHome(value): This feature allows you to specify the directory where Spark will be installed on worker nodes.

setAppName(value): This element is used to specify the name of the application.

setMaster(value): The master URL may be set using this property.

get(key, defaultValue=None): This attribute aids in the retrieval of a key's configuration value.

#### Q3. What is the key difference between list and tuple?

The primary difference between lists and tuples is that lists are mutable, but tuples are immutable.

When a Python object may be edited, it is considered to be a mutable data type. Immutable data types, on the other hand, cannot be changed.

Here’s an example of how to change an item list into a tuple-

list\_num[3] = 7

print(list\_num)

tup\_num[3] = 7

Output:

[1,2,5,7]

Traceback (most recent call last):

File "python", line 6, in

TypeError: 'tuple' object doesnot support item assignment

We assigned 7 to list\_num at index 3 in this code, and 7 is found at index 3 in the output. However, we set 7 to tup\_num at index 3, but the result returned a type error. Because of their immutable nature, we can't change tuples.

#### Q4. What do you understand by errors and exceptions in Python?

There are two types of errors in Python: syntax errors and exceptions.

Syntax errors are frequently referred to as parsing errors. Errors are flaws in a program that might cause it to crash or terminate unexpectedly. When a parser detects an error, it repeats the offending line and then shows an arrow pointing to the line's beginning.

Exceptions arise in a program when the usual flow of the program is disrupted by an external event. Even if the program's syntax is accurate, there is a potential that an error will be detected during execution; nevertheless, this error is an exception. ZeroDivisionError, TypeError, and NameError are some instances of exceptions.

#### Q5. What are the most significant changes between the Python API (PySpark) and Apache Spark?

PySpark is a Python API created and distributed by the Apache Spark organization to make working with Spark easier for Python programmers. Scala is the programming language used by Apache Spark. It can communicate with other languages like Java, R, and Python.

Also, because Scala is a compile-time, type-safe language, Apache Spark has several capabilities that PySpark does not, one of which includes Datasets. Datasets are a highly typed collection of domain-specific objects that may be used to execute concurrent calculations.

#### Q6. Define SparkSession in PySpark. Write code to create SparkSession in PySpark

Spark 2.0 includes a new class called SparkSession (pyspark.sql import SparkSession). Prior to the 2.0 release, SparkSession was a unified class for all of the many contexts we had (SQLContext and HiveContext, etc). Since version 2.0, SparkSession may replace SQLContext, HiveContext, and other contexts specified before version 2.0. It's a way to get into the core PySpark technology and construct PySpark RDDs and DataFrames programmatically. Spark is the default object in pyspark-shell, and it may be generated programmatically with SparkSession.

In PySpark, we must use the builder pattern function builder() to construct SparkSession programmatically (in a.py file), as detailed below. The getOrCreate() function retrieves an already existing SparkSession or creates a new SparkSession if none exists.

spark=SparkSession.builder.master("local[1]") \

                    .appName('ProjectPro') \

                    .getOrCreate()

#### Q7. Suppose you encounter the following error message while running PySpark commands on Linux-

#### ImportError: No module named py4j.java\_gateway

#### How will you resolve it?

Py4J is a Java library integrated into PySpark that allows Python to actively communicate with JVM instances. Py4J is a necessary module for the PySpark application to execute, and it may be found in the $SPARK\_HOME/python/lib/py4j-\*-src.zip directory.

To execute the PySpark application after installing Spark, set the Py4j module to the PYTHONPATH environment variable. We’ll get an ImportError: No module named py4j.java\_gateway error if we don't set this module to env.

So, here’s how this error can be resolved-

export SPARK\_HOME=/Users/abc/apps/spark-3.0.0-bin-hadoop2.7

export PYTHONPATH=$SPARK\_HOME/python:$SPARK\_HOME/python/build:$SPARK\_HOME/python/lib/py4j-0.10.9-src.zip:$PYTHONPATH

Put these in .bashrc file and re-load it using source ~/.bashrc

The py4j module version changes depending on the PySpark version we’re using; to configure this version correctly, follow the steps below:

export PYTHONPATH=${SPARK\_HOME}/python/:$(echo ${SPARK\_HOME}/python/lib/py4j-\*-src.zip):${PYTHONPATH}

Use the pip show command to see the PySpark location's path-  pip show pyspark

Use the environment variables listed below to fix the problem on Windows-

set SPARK\_HOME=C:\apps\opt\spark-3.0.0-bin-hadoop2.7

set HADOOP\_HOME=%SPARK\_HOME%

set PYTHONPATH=%SPARK\_HOME%/python;%SPARK\_HOME%/python/lib/py4j-0.10.9-src.zip;%PYTHONPATH%

#### Q8. Suppose you get an error- NameError: Name 'Spark' is not Defined while using spark. createDataFrame(), but there are no errors while using the same in Spark or PySpark shell. Why?

Spark shell, PySpark shell, and Databricks all have the SparkSession object 'spark' by default. However, if we are creating a Spark/PySpark application in a.py file, we must manually create a SparkSession object by using builder to resolve NameError: Name 'Spark' is not Defined.

# Import PySpark

import pyspark

from pyspark.sql import SparkSession

#Create SparkSession

spark = SparkSession.builder

                    .master("local[1]")

                    .appName("SparkByExamples.com")

                    .getOrCreate()

If you get the error message 'No module named pyspark', try using findspark instead-

#Install findspark

pip install findspark

# Import findspark

import findspark

findspark.init()

#import pyspark

import pyspark

from pyspark.sql import SparkSession

#### Q9. What are the various types of Cluster Managers in PySpark?

Spark supports the following [cluster managers](https://www.projectpro.io/article/apache-spark-architecture-explained-in-detail/338" \o "cluster managers" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank):

Standalone- a simple cluster manager that comes with Spark and makes setting up a cluster easier.

Apache Mesos- Mesos is a cluster manager that can also run Hadoop MapReduce and PySpark applications.

Hadoop YARN- It is the Hadoop 2 resource management.

Kubernetes- an open-source framework for automating containerized application deployment, scaling, and administration.

local – not exactly a cluster manager, but it's worth mentioning because we use "local" for master() to run Spark on our laptop/computer.

#### Q10. Explain how Apache Spark Streaming works with receivers.

Receivers are unique objects in Apache Spark Streaming whose sole purpose is to consume data from various data sources and then move it to Spark. By streaming contexts as long-running tasks on various executors, we can generate receiver objects.

There are two different kinds of receivers which are as follows:

Reliable receiver: When data is received and copied properly in Apache Spark Storage, this receiver validates data sources.

Unreliable receiver: When receiving or replicating data in Apache Spark Storage, these receivers do not recognize data sources.

You can refer to [GitHub](https://github.com/spark-examples/pyspark-examples" \o "GitHub" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) for some of the examples used in this blog.

Become a data engineer and put your skills to the test! But the problem is, where do you start? Broadening your expertise while focusing on an advanced understanding of certain technologies or languages is a good idea. There is no better way to learn all of the necessary big data skills for the job than to do it yourself. [ProjectPro](https://www.projectpro.io/projects" \o "ProjectPro" \t "https://www.projectpro.io/article/pyspark-interview-questions-and-answers/_blank) provides a customised learning path with a variety of completed big data and data science projects to assist you in starting your career as a data engineer.

## ****FAQs****

### ****1. Is PySpark the same as Spark?****

No. PySpark is Python API for Spark. PySpark allows you to create applications using Python APIs.

****2. What is PySpark, and how it works?****

PySpark is a Python API for Apache Spark. It lets you develop Spark applications using Python APIs, but it also includes the PySpark shell, which allows you to analyze data in a distributed environment interactively. Most of Spark's capabilities, such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning), and Spark Core, are supported by PySpark.

### ****3. Is PySpark a Big Data tool? Does PySpark require Spark?****

Yes, PySpark is a faster and more efficient Big Data tool.

PySpark is a Python Spark library for running Python applications with Apache Spark features. Hence, it cannot exist without Spark.

### ****4. Is PySpark easy to learn?****

PySpark is easy to learn for those with basic knowledge of Python, Java, etc.

### ****5. How long does it take to learn PySpark?****

One week is sufficient to learn the basics of the Spark Core API if you have significant knowledge of object-oriented programming and functional programming.

### ****6. What is the best way to learn PySpark? Is PySpark a framework?****

PySpark is an open-source framework that provides Python API for Spark.

## **Benefits of Using PySpark**

PySpark is a big data framework tool for Python that helps to perform large-scale data processing, analysis, and manipulation. The tool offers many benefits to developers and data professionals, as it is equipped with an extensive library with functions capable of performing a wide range of tasks.

Some notable benefits of using PySpark include:

1. PySpark supports In-memory processing, a feature that rapidly improves the speed of processing large data sets
2. PySpark has over 80 operators, helping data analysts to perform a wide range of data-related tasks
3. PySpark is dynamic and offers third party integration
4. Many essential algorithms are integrated within the framework, offering users the ease to carry out complex tasks
5. Programmers and data professionals can easily handle and manage errors
6. PySpark's huge set of libraries makes it possible for developers to perform complex data processing operations

## **PySpark Interview Questions for Freshers**

If you’re a fresher applying for a Python developer or data analyst role, being able to answer common PySpark interview questions is essential from an interview perspective.

To give you an idea of the type of questions asked, below are some common PySpark interview questions.

1. What are the main characteristics of the PySpark framework?
2. What is SparkConf in PySpark?
3. What do you understand about SparkFiles in PySpark?
4. How do you get the absolute path of a file in PySpark?
5. Which function is used to retrieve the root directory in PySpark?
6. Explain the different storage levels in PySpark.
7. What are broadcast variables in PySpark?
8. What is Martial and Pickel Serializers in the PySpark framework?
9. How do you get information about Spark Jobs in PySpark?
10. What are filters in PySpark?
11. Explain Accumulator variables in PySpark.
12. Explain the SparkStageInfo in PySpark.

## **PySpark Interview Questions for Experienced Professionals**

Experienced Python programmers, developers, data engineers, data scientists, and analysts are expected to know their way around PySpark. If you’re an experienced tech professional, you can expect quite a few PySpark interview questions for experienced professionals at tech interviews.

Here are the top questions to give you context about the questions to expect.

1. Which are some algorithms supported and integrated into PySpark?
2. How do you create a UDF in PySpark?
3. What do you understand about the PySpark DAGScheduler?
4. What do you understand by shared variables in PySpark?
5. What are DataFrames in PySpark? How are they different from DataFrames in Pandas?
6. How do you create a SparkSession in PySpark?
7. How do you create a Resilient Distributed DataSet in PySpark? What are the approaches you can take?
8. How do you join two DataFrames in PySpark?
9. How do you go about performing streaming in PySpark?
10. What do you understand about profilers in PySpark?

Practicing these above PySpark interview questions can help you prepare for your upcoming technical interview. For specific coding-related problems for technical interviews, check out our **[Problems Page](https://www.interviewkickstart.com/problems)**.

## **FAQs on PySpark Interview Questions**

****Q1. What is PySpark?****

PySpark is an open-source, cluster-computing framework integrated into Apache Spark. It helps to perform large-scale data analysis and processing by offering a wide set of libraries that help developers and analysts to perform complex data-related functions.

****Q2. Why is it important to practice PySpark interview questions?****

Practicing PySpark interview questions is crucial if you’re appearing for a Python, data engineering, data analyst, or data science interview, as companies often expect you to know your way around powerful data-processing tools and frameworks (like PySpark).

****Q3. What roles require a good understanding and knowledge of PySpark?****

Roles that require a good understanding of PySpark include - data engineering, data scientist, data analyst, big-data, and Python developer.

****Q4. What are some benefits of PySpark?****

Some benefits of PySpark are - i) it offers a huge set of libraries, ii) it can be integrated with 3rd party applications, iii) it allows for swift data processing, up to 10x faster than traditional frameworks iv) it is easy to use.

****Q5. What are some industrial benefits of using PySpark?****

Some industrial applications of PySpark include - Real-time media streaming, Financial analysis, processing healthcare information and analyzing healthcare records, in-app travel and e-commerce recommendations, and telecommunication services.

**Que 1. Explain PySpark in brief?**

Ans. As Spark is written in Scala so in order to support Python with Spark, Spark Community released a tool, which we call PySpark. In Python programming language, we can also work with RDDs, using PySpark. It is possible due to its library name Py4j.

**Que 2. What are the main characteristics of (Py)Spark?**

Ans. Some of  the main characteristics of (Py)Spark are:

* Here Nodes are abstracted that says no possible to address an individual node.
* Also, Network is abstracted, that means there is only implicit communication possible.
* Moreover, it is based on Map-Reduce, that means programmer provides a map and a reduce function here.
* And, PySpark is one of the API for Spark.

**Que 3. Pros of PySpark?**

Ans. Some of the benefits of using PySpark are:

* For simple problems, it is very simple to write parallelized code.
* Also, it handles Synchronization points as well as errors.
* Moreover, in Spark, many useful algorithms is already implemented.

**Que 4. Cons of PySpark?**

Ans.   Some of the limitations on using PySpark are:

* It is difficult to express a problem in MapReduce fashion sometimes.
* Also, Sometimes, it is not as efficient as other programming models.

**Que 5. Prerequisites to learn PySpark?**

Ans. It is being assumed that the readers are already aware of what a programming language and a framework is, before proceeding with the various concepts given in this tutorial. Also, if the readers have some knowledge of Spark and Python in advance, it will be very helpful.

**Que 6. What do you mean by PySpark SparkContext?**

Ans. In simple words, an entry point to any spark functionality is what we call SparkContext. While it comes to ****PySpark, SparkContext**** uses Py4J(library) in order to launch a JVM. In this way, it creates a JavaSparkContext. However, PySpark has SparkContext available as ‘sc’, by default.

**Que 7. Explain PySpark SparkConf?**

Ans. Mainly, we use ****SparkConf**** because we need to set a few configurations and parameters to run a Spark application on the local/cluster. In other words, SparkConf offers configurations to run a Spark application.

* Code

**class** pyspark.SparkConf (

loadDefaults = **True**,

\_jvm = None,

\_jconf = None

)

**Que 8. Tell us something about PySpark SparkFiles?**

Ans. It is possible to upload our files in Apache Spark. We do it by using sc.addFile, where sc is our default SparkContext. Also, it helps to get the path on a worker using SparkFiles.get. Moreover, it resolves the paths to files which are added through SparkContext.addFile().

It contains some classmethods, such as −

* get(filename)
* getrootdirectory()

**Que 9. Explain get(filename).**

Ans.  It helps to get the absolute path of a file, which are added through SparkContext.addFile().

**def** get(cls, filename):

path = os.path.join(SparkFiles.getRootDirectory(), filename)

**return** os.path.abspath(path)

**Que 10. Explain getrootdirectory().**

Ans. Whereas, it helps to get the root directory which is consist of the files which are added through SparkContext.addFile().

**def** getRootDirectory(cls):

**if** cls.\_is\_running\_on\_worker:

**return** cls.\_root\_directory

**else**:

# This will have to change if we support multiple SparkContexts:

**return** cls.\_sc.\_jvm.org.apache.spark.SparkFiles.getRootDirectory()

****PySpark Interview Questions for freshers – Q. 1,2,3,4,5,6,7,8****

****PySpark Interview Questions for experienced – Q. 9,10****

**Que 11. Explain PySpark StorageLevel in brief.**

Ans. Basically, it controls that how an RDD should be stored. Also, it controls if to store  RDD in the memory or over the disk, or both. In addition, even it controls that we need to serialize RDD or to replicate RDD partitions.

* Code

**class** pyspark.StorageLevel(useDisk, useMemory, useOffHeap, deserialized, replication = 1)

**Que 12. Name different storage levels.**

Ans. There are different ****storage levels****, which are given below −

* DISK\_ONLY StorageLevel(True, False, False, False, 1)
* DISK\_ONLY\_2 StorageLevel(True, False, False, False, 2)
* MEMORY\_AND\_DISK StorageLevel(True, True, False, False, 1)
* MEMORY\_AND\_DISK\_2 StorageLevel(True, True, False, False, 2)
* MEMORY\_AND\_DISK\_SER StorageLevel(True, True, False, False, 1)
* MEMORY\_AND\_DISK\_SER\_2 StorageLevel(True, True, False, False, 2)
* MEMORY\_ONLY StorageLevel(False, True, False, False, 1)
* MEMORY\_ONLY\_2StorageLevel(False, True, False, False, 2)
* MEMORY\_ONLY\_SER StorageLevel(False, True, False, False, 1)
* MEMORY\_ONLY\_SER\_2  StorageLevel(False, True, False, False, 2)
* OFF\_HEAP  StorageLevel(True, True, True, False, 1)

**Que 13. What do mean by Broadcast variables?**

Ans. In order to save the copy of data across all nodes, we use it.   
With SparkContext.broadcast(), a broadcast variable is created.   
For Examples:

>>> from pyspark.context import SparkContext

>>> sc = SparkContext('local', 'test')

>>> b = sc.broadcast([1, 2, 3, 4, 5])

>>> b.value

[1, 2, 3, 4, 5]

>>> sc.parallelize([0, 0]).flatMap(lambda x: b.value).collect()

[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

>>> b.unpersist()

>>> large\_broadcast = sc.broadcast(range(10000))

**Que 14. What are Accumulator variables?**

Ans. In order to aggregate the information through associative and commutative operations, we use them.

* Code

**class** pyspark.Accumulator(aid, value, accum\_param)

**Que 15. Explain AccumulatorParam?**

Ans. AccumulatorParam is a helper object which explains how to accumulate values of a given type.  
**class AccumulatorParam(object):**  
**def zero(self, value):**  
**“””**  
**Also,**  
****with the provided C{value} (e.g., a zero vector) it****  
****Provides a “zero value” for the type, compatible in dimensions****  
**“””**  
**raise NotImplementedError**  
**def addInPlace(self, value1, value2):**

**Que 16. Why we need Serializers in PySpark?**

Ans. For the purpose of performance tuning, PySpark supports custom serializers, such as−

* MarshalSerializer
* PickleSerializer

**Que 17. Explain Marshal Serializer?**

Ans. With the help of Python’s Marshal Serializer, it serializes objects. Even if it supports fewer datatypes, it is faster than PickleSerializer.

**class** MarshalSerializer(FramedSerializer):

**def** dumps(self, obj):

**return** marshal.dumps(obj)

**def** loads(self, obj):

**return** marshal.loads(obj)

**Que 18. Explain Pickel Serializers?**

Ans.  This uses Python’s Pickle Serializer to serialize objects. It supports nearly any Python object, but in slow speed.

**class** PickleSerializer(FramedSerializer):

**def** dumps(self, obj):

**return** pickle.dumps(obj, protocol)

**if** sys.version >= '3':

**def** loads(self, obj, encoding="bytes"):

**return** pickle.loads(obj, encoding=encoding)

**else**:

**def** loads(self, obj, encoding=None):

**return** pickle.loads(obj)

**Que 19. What do you mean by Status Tracker?**

Ans. Status Trackers are Low-level status reporting APIs which helps to monitor job and stage progress.

**def** \_\_init\_\_(self, jtracker):

self.\_jtracker = jtracker

****Que 20. Explain SparkJobinfo?****

Ans. SparkJobinfo exposes information about ****Spark Jobs****.

**class** SparkJobInfo(namedtuple("SparkJobInfo", "jobId stageIds status")):

****PySpark Interview Questions for freshers – Q. 11,12,13,14,16,17,18,19****

****PySpark Interview Questions for experienced – Q. 15,20****

**Que 21. Explain SparkStageinfo?**

Ans. SparkStageinfo exposes information about Spark Stages

**class** SparkStageInfo(namedtuple("SparkStageInfo",

"stageId currentAttemptId name numTasks numActiveTasks "

"numCompletedTasks numFailedTasks")):

****Que 22. Which Profilers do we use in PySpark?****

Ans.  Custom profilers are PySpark supported in PySpark to allow for different ****Profilers**** to be used an for outputting to different formats than what is offered in the BasicProfiler.  
We need to define or inherit the following methods, with a custom profiler:

* ****profile**** – Basically, it produces a system profile of some sort.
* ****stats**** – Well, it returns the collected stats.
* ****dump**** – Whereas, it dumps the profiles to a path.
* ****add**** – Moreover, this method helps to add a profile to the existing accumulated profile

Generally, when we create a SparkContext, we choose the profiler class.

**Que 23. Explain Basic Profiler.**

Ans. It is a default profiler, which we implement on the basis of cProfile and Accumulator.

**Que 24. Do, we have machine learning API in Python?**

Ans. As Spark provides a ****Machine Learning**** API, MLlib. Similarly, in ****Python**** as well, PySpark has this machine learning API.

**Que 25. Name algorithms supported in PySpark?**

Ans. There are several ****algorithms in PySpark****:

* mllib.classification
* mllib.clustering
* mllib.fpm
* mllib.linalg
* mllib.recommendation
* spark.mllib
* Mllib.regression

**Que 26. Name parameter of SparkContext?**

Ans. The parameters of a SparkContext are:

* ****Master**** − URL of the cluster from which it connects.
* ****appName**** − Name of our job.
* ****sparkHome**** − Spark installation directory.
* ****pyFiles**** − It is the .zip or .py files, in order to send to the cluster and also to add to the PYTHONPATH.
* ****Environment**** − Worker nodes environment variables.
* ****Serializer**** − RDD serializer.
* ****Conf**** − to set all the Spark properties, an object of L{SparkConf}.
* ****JSC**** − It is the JavaSparkContext instance.

**Que 27. Which of the parameters of SparkContext we mostly use?**

Ans. Master and app name.

**Que 28. Name attributes of SparkConf.**

Ans. ****Attributes of SparkConf**** −

1. set(key, value) − This attribute helps to set a configuration property.
2. setMaster(value) − It helps to set the master URL.
3. setAppName(value) − This helps to set an application name.
4. get(key, defaultValue=None) − This attribute helps to get a configuration value of a key.
5. setSparkHome(value) − It helps to set Spark installation path on worker nodes.

****Que 29. Why Profiler?****

Ans. Profilers help us to ensure that the applications do not waste any resources also to spot any problematic code.

**Que 30. State Key Differences in the Python API.**

Ans.  Differences between the Python and Scala APIs are:

* It is dynamically typed hence because of that RDDs can hold objects of multiple types.
* On comparing with Scala, PySpark does not yet support some APIs.

Attending a PySpark interview and wondering what are all the questions and discussions you will go through? Before attending a PySpark interview, it’s better to have an idea about the types of PySpark interview questions that will be asked so that you can mentally prepare answers for them.

To help you out, I have created the top PySpark interview question and answers guide to understand the depth and real-intend of PySpark interview questions. Let’s get started.

As the name suggests, PySpark is an integration of Apache Spark and the Python programming language. Apache Spark is a widely used open-source framework that is used for cluster-computing and is developed to provide an easy-to-use and faster experience. Python is a high-level general-purpose programming language. It is mainly used for Data Science, Machine Learning and Real-Time Streaming Analytics, apart from its many other uses.

Originally, Apache spark is written in the Scala programming language, and PySpark is actually the Python API for Apache Spark. In this article, we will take a glance at the most frequently asked PySpark interview questions and their answers to help you get prepared for your next interview. If you are a beginner and interested to learn more about data science, check out our data analytics certification from top universities.

****Read:****[Dataframe in Apache PySpark](https://www.upgrad.com/blog/dataframe-in-apache-pyspark-tutorial-examples/)



[Source](https://medium.com/@gulcanogundur/pyspark-word-count-b099106135a7" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank)

**Table of Contents**

## ****PySpark Interview Questions and Answers****

****1. What is PySpark?****

This is almost always the first PySpark interview question you will face.

PySpark is the Python API for Spark. It is used to provide collaboration between Spark and Python. PySpark focuses on processing structured and semi-structured data sets and also provides the facility to read data from multiple sources which have different data formats. Along with these features, we can also interface with RDDs (Resilient Distributed Datasets ) using PySpark. All these features are implemented using the py4j library.

****2. List the advantages and disadvantages of PySpark?**** (Frequently asked ****PySpark Interview Question****)

***The advantages of using PySpark are:***

* Using the PySpark, we can write a parallelized code in a very simple way.
* All the nodes and networks are abstracted.
* PySpark handles all the errors as well as synchronization errors.
* PySpark contains many useful in-built algorithms.

**Must read**: [Learn excel online free](https://www.upgrad.com/blog/excel-free-online-course/)!

***The disadvantages of using PySpark are:***

* PySpark can often make it difficult to express problems in MapReduce fashion.
* When compared with other programming languages, PySpark is not efficient.

## **Explore our Popular Data Science Courses**

|  |  |  |
| --- | --- | --- |
| [Executive Post Graduate Programme in Data Science from IIITB](https://www.upgrad.com/data-science-pgd-iiitb/?utm_source=BLOG&utm_medium=TABLE&utm_campaign=DV_DA_PGD_BLOG_TABLE" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Professional Certificate Program in Data Science for Business Decision Making](https://www.upgrad.com/data-science-professional-certificate-bdm-iimk/?utm_source=BLOG&utm_medium=TABLE&utm_campaign=DV_DA_IIMK_BLOG_TABLE" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Master of Science in Data Science from University of Arizona](https://www.upgrad.com/data-science-ms-uoa/?utm_source=BLOG&utm_medium=TABLE&utm_campaign=DV_DA_UAO_BLOG_TABLE" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) |
| [Advanced Certificate Programme in Data Science from IIITB](https://www.upgrad.com/data-science-pgc-iiitb/?utm_source=BLOG&utm_medium=TOP_MENUBAR&utm_campaign=DV_DA_PGC_BLOG_TOP_MENUBAR" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Professional Certificate Program in Data Science and Business Analytics from University of Maryland](https://www.upgrad.com/executive-pgp-in-data-science-business-analytics-maryland/?utm_source=BLOG&utm_medium=TABLE&utm_campaign=DV_DA_CERT_BLOG_TABLE" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Data Science Courses](https://www.upgrad.com/data-science-course/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_self) |

****3. What are the various algorithms supported in PySpark?****

***The different algorithms supported by PySpark are:***

1. spark.mllib
2. mllib.clustering
3. mllib.classification
4. mllib.regression
5. mllib.recommendation
6. mllib.linalg
7. mllib.fpm

****4. What is PySpark SparkContext?****

PySpark SparkContext can be seen as the initial point for entering and using any Spark functionality. The SparkContext uses py4j library to launch the JVM, and then create the JavaSparkContext. By default, the SparkContext is available as ‘sc’.

****5. What is PySpark SparkFiles?****

One of the most common PySpark interview questions. PySpark SparkFiles is used to load our files on the Apache Spark application. It is one of the functions under SparkContext and can be called using sc.addFile to load the files on the Apache Spark. SparkFIles can also be used to get the path using SparkFile.get or resolve the paths to files that were added from sc.addFile. The class methods present in the SparkFiles directory are getrootdirectory() and get(filename).

****Read:**** [Spark Project Ideas](https://www.upgrad.com/blog/spark-project-ideas-topics-for-beginners/)

*****upGrad’s Exclusive Data Science Webinar for you –*****

Video Player

00:00

35:20

****6. What is PySpark SparkConf?****

PySpark SparkConf is mainly used to set the configurations and the parameters when we want to run the application on the local or the cluster.  
We run the following code whenever we want to run SparkConf:

class pyspark.Sparkconf(

localdefaults = True,

\_jvm = None,

\_jconf = None

)

****7. What is PySpark StorageLevel?****

PySpark StorageLevel is used to control how the RDD is stored, take decisions on where the RDD will be stored (on memory or over the disk or both), and whether we need to replicate the RDD partitions or to serialize the RDD. The code for StorageLevel is as follows:

class pyspark.StorageLevel( useDisk, useMemory, useOfHeap, deserialized, replication = 1)

****8. What is PySpark SparkJobinfo?****

One of the most common questions in any PySpark interview. PySpark SparkJobinfo is used to gain information about the SparkJobs that are in execution. The code for using the SparkJobInfo is as follows:

class SparkJobInfo(namedtuple(“SparkJobInfo”, “jobId stageIds status ”)):



## **Read our popular Data Science Articles**

|  |  |  |
| --- | --- | --- |
| [Data Science Career Path: A Comprehensive Career Guide](https://www.upgrad.com/blog/data-science-learning-path/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Data Science Career Growth: The Future of Work is here](https://www.upgrad.com/blog/data-science-career-growth/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Why is Data Science Important? 8 Ways Data Science Brings Value to the Business](https://www.upgrad.com/blog/why-data-science-important/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) |
| [Relevance of Data Science for Managers](https://www.upgrad.com/blog/data-science-for-managers/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [The Ultimate Data Science Cheat Sheet Every Data Scientists Should Have](https://www.upgrad.com/blog/data-science-cheat-sheet/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Top 6 Reasons Why You Should Become a Data Scientist](https://www.upgrad.com/blog/reasons-to-become-a-data-scientist/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) |
| [A Day in the Life of Data Scientist: What do they do?](https://www.upgrad.com/blog/a-day-in-the-life-of-data-scientist/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Myth Busted: Data Science doesn’t need Coding](https://www.upgrad.com/blog/myth-busted-data-science-doesnt-need-coding/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Business Intelligence vs Data Science: What are the differences?](https://www.upgrad.com/blog/data-science-vs-bi/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) |

****9. What is PySpark SparkStageinfo?****

One of the most common question in any PySpark interview question and answers guide. PySpark SparkStageInfo is used to gain information about the SparkStages that are present at that time. The code used fo SparkStageInfo is as follows:

class SparkStageInfo(namedtuple(“SparkStageInfo”, “stageId currentAttemptId name numTasks unumActiveTasks” “numCompletedTasks numFailedTasks” )):

**Our learners also read:** [Free Python Course with Certification](https://www.upgrad.com/blog/python-free-online-course/)

****10. What is PySpark DataFrames?****

This is one of the most common**PySpark dataframe interview questions.**PySpark DataFrames are the distributed assortment of well-organized data. They are identical to relational database tables and are included in named columns. Moreover, PySpark DataFrames are more efficiently optimized than Python or R programming languages. The reason is they can be created from various sources like Structured Data Files, Hive Tables, external databases, existing RDDs, etc.

The greatest advantage of using PySpark DataFrame is that the data in it is distributed over various machines in the cluster. The corresponding operations will run parallel on all the machines.

## **Top Data Science Skills to Learn in 2022**

|  |  |  |
| --- | --- | --- |
| **SL. No** | **Top Data Science Skills to Learn in 2022** | |
| 1 | [Data Analysis Course](https://www.upgrad.com/data-science-course/data-analysis/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_blank) | [Inferential Statistics Courses](https://www.upgrad.com/data-science-course/inferential-statistics/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_self) |
| 2 | [Hypothesis Testing Programs](https://www.upgrad.com/data-science-course/hypothesis-testing/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_self) | [Logistic Regression Courses](https://www.upgrad.com/data-science-course/logistic-regression/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_self) |
| 3 | [Linear Regression Courses](https://www.upgrad.com/data-science-course/linear-regression/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_self) | [Linear Algebra for Analysis](https://www.upgrad.com/data-science-course/linear-algebra-for-analysis/" \t "https://www.upgrad.com/blog/pyspark-interview-questions-answers/_self) |

****11. What is PySpark Join?****

PySpark Join helps combine two DataFrames. By binding these, it is easy to join multiple DataFrames. It enables all fundamental join type operations accessible in traditional SQL like INNER, RIGHT OUTER, LEFT OUTER, LEFT SEMI, LEFT ANTI, SELF JOIN, and CROSS. PySpark Joins are transformations that use data shuffling throughout the network.

****12. How to rename a DataFrame column in PySpark?****

It is one of the most frequently asked **PySpark dataframe interview questions.**You can use PySpark withColumnRenamed() to rename a DataFrame column. Frequently, you need to remain single or multiple columns on PySpark DataFrame. It can be done in multiple ways. DataFrame is an immutable collection, so you can’t update or rename a column instead when using withColumnRenamed(). This is because it prepares a new DataFrame with the updated column names. Two common ways to rename nested columns are –renaming all columns or renaming selected multiple columns.

****13. Are PySpark and Spark the same?****

These types of **PySpark coding questions**test the candidates’ basic knowledge of the PySpark fundamentals. PySpark has been launched to support the collaboration of Python and Apache Spark. Essentially, it is a Python API for Spark. PySpark assists you in interfacing with Resilient Distributed Datasets (RDDs) in Python programming language and Apache Spark.

****14. What is PySparkSQL?****

When preparing for**PySpark coding interview questions,**you must prepare for PySparkSQL. It is a PySpark library to implement SQL-like analysis on a large amount of either structured or semi-structured data. You can also use SQL queries with PySparkSQL. Moreover, it can be connected to Apache Hive, and HiveQL can also be implemented.

PySparkSQL works as a wrapper over the PySpark core. PySparkSQL introduced the DataFrame, a tabular illustration of structured data that is identical to that of a table from an RDBMS (relational database management system).

****15. Are there any prerequisites to learning PySpark?****

One of the fundamental **PySpark coding questions**is about the prerequisites to learn PySpark. It is assumed that the readers are aware of what a framework and a programming language are before moving towards different concepts in the PySpark tutorial. It is beneficial if the readers have some knowledge of Python and Spark in advance.

****16. What do you understand by PySpark SparkFiles?****

It is allowed to upload our files in Apache Spark by using sc.addFile. Here sc is the default SparkContext. It also assists in getting the path on a worker through SparkFiles.get. It also resolves the paths to files that are added via SparkContext.addFile().PySpark SparkFiles includes certain classmethods likeget(filename) and getrootdirectory().

****17. What are the key characteristics of PySpark?****

Knowing PySpark characteristics is important after you complete preparing for the **PySpark coding interview questions.**The four key characteristics of PySpark are as below. (i) Nodes are abstracted: You can’t access the individual worker nodes. (ii) APIs for Spark features: PySpark offers APIs for using Spark features. (iii) PySpark is dependent on MapReduce: PySpark is dependent on the MapReduce model of Hadoop. So, it lets a programmer provide the map and the reduced functions. (iv) Abstracted Network: Abstracted networks in PySpark allow implicit communication only.

****18. What is SparkCore? What are the major functions of SparkCore?****

SparkCore is the Spark platform’s general execution engine that supports all the functionalities. It provides in-memory computing capabilities to offer a decent speed and a universal execution model to support different applications. It also supports Scala, Java, and Python APIs to simplify the development process. The key functions of SparkCore include the basic I/O functions, monitoring, scheduling, effective memory management, fault tolerance, fault recovery, and interaction with storage systems.

****19. What it means by PySpark serializers?****

One of the mid-level**PySpark interview coding questions** can be around PySpark serializers. In PySpark, the serialization process is used to perform Spark performance tuning. PySpark incorporates serializers because you must constantly check the data sent or received across the network to the memory or disk. Two types of serializers in PySpark are as below. (i) PickleSerializer: It serializes the objects using Python’s PickleSerializer and class pyspark.PickleSerializer). It supports most of the Python objects. (ii) MarshalSerializer: It performs objects’ serialization. It can be employed through class pyspark.MarshalSerializer. It is faster than the PickleSerializer, but it supports limited types.

****20. What is PySpark ArrayType?****

PySpark ArrayType is a collection data type that outspreads PySpark’s DataType class (the superclass for all types). It only contains the same types of files. You can use ArraType()to construct an instance of an ArrayType. Two arguments it accepts are discussed below. (i) valueType: The valueType must extend the DataType class in PySpark. (ii) valueContainsNull: It is an optional argument that states whether a value can accept null and it is by default value, is True.

****21. What is PySpark Partition? How many partitions can one make in PySpark?****

You may be asked a **PySpark interview question**around PySpark Partition. It is a method that splits a huge dataset into smaller datasets depending on one or multiple partition keys. It improves the execution speed when the transformations on partitioned data operate faster. The reason is that every partition’s transformations run in parallel. PySpark allows two types of partitioning i.e. partitioning on disc (File system) and partitioning in memory (DataFrame). Its syntax is partitionBy (self, \*cols) . Including 4x of partitions to the number of cores in the cluster accessible for application is recommended.

****22. What is Parquet file in PySpark?****

You may be asked **PySpark interview coding questions** on the file type in PySpark. The Parquet file in PySpark is a column-type format supported by different data processing systems. It helps Spark SQL to perform read and write operations. Its column-type format storage offers the following benefits. (i) It consumes less space. (ii)It allows you to retrieve specific columns for access. (iii)It employs type-specific encoding. (iv)It provides better-summarized data. (v)It supports limited I/O operations.

****23. Why is PySpark faster than pandas?****

This kind of **PySpark interview question**tests your in-depth knowledge of PySpark. PySpark is speedier than pandas because it supports parallel execution of statements in a distributed environment. PySpark can be implemented on different machines and cores not supported in Pandas.

## ****Conclusion****

We hope you went through all the frequently asked ****PySpark Interview Questions****. Apache Spark is mainly used to handle BigData and is in very high demand as companies move forward to use the latest technologies to drive their businesses.

If you’re interested to learn python & want to get your hands dirty on various tools and libraries, check out Executive PG Program in Data Science.

If you are interested to know more about Big Data, check out our [Advanced Certificate Programme in Big Data from IIIT Bangalore](https://www.upgrad.com/big-data-certification-pgc-iiitb/?utm_source=BLOG&utm_medium=BODY&utm_campaign=TV_SEBE_PGC_BLOG_BODY_87336). Do check out his course in order to learn from the best academicians and industry leaders to upgrade your career in this field.

Study[data science courses](https://www.upgrad.com/data-science-course/?utm_source=BLOG&utm_medium=BODY&utm_campaign=DV_DA_BLOG_BODY_VP_87336) online from the World’s top Universities. Earn Executive PG Programs, Advanced Certificate Programs, or Masters Programs to fast-track your career.

## **What is Cluster Computing?**

Cluster Computing consists of loosely coupled systems that interact, work, and perform operations as a single system. The various cluster nodes are connected via LAN (Local Area Network). Cluster computing ensures scalability, speed, resource management, and continuous availability of computing power. Clusters are of two types: Open and closed. Open clusters are those through which nodes can be accessed only via the Internet. In closed clusters, the nodes are hidden and secure. Each cluster computer consists of cluster nodes, cluster operating system, switches, and network-switching hardware.

## **What is the average salary of an Apache PySpark Developer in India?**

A PySpark developer ensures that data is available for query processing. An Apache PySpark developer should be good at Python, Apache Spark, Java, and Scala. The demand for Apache Spark developers has been increasing. One can get more than 60000 search results of job opportunities for these roles. The salary, however, depends on many factors. These include work experience, skill set, demand in the market, organisation, location, etc. Based on these, the salary could range from INR 8 LPA to INR 20 LPA. The average wages for people with less than two years of experience range from INR 4.5 LPA to INR 15.7 LPA.

## **What is meant by RDD?**

RDD stands for Resilient Distributed Dataset (RDD). It is a data structure that stores immutable objects. It supports the storage of objects of any language, like Python, Java, Scala, and other user-defined objects. MapReduce is used for massively parallel processing of data quickly. Spark uses RDD to perform MapReduce operations. RDDs can be created in 2 ways: either by parallelising a data set in your system or by referencing an external data storage system. RDD is fault-tolerant and supports parallel processing. It is mainly used to process and manipulate unstructured data. RDD is a distributed system. It follows the Lazy Evaluation Principle, i.e. transformations are applied only when we call it and not when the data is loaded.

<https://www.tutorialspoint.com/pyspark/index.htm>

https://www.javatpoint.com/pyspark