**1).What is Apache Spark?**

Spark is a general-purpose distributed data processing engine that is suitable for use in a wide range of circumstances. On top of the Spark core data processing engine, there are libraries for SQL, machine learning, graph computation, and stream processing, which can be used together in an application. Programming languages supported by Spark include: Java, Python, Scala, and R. Application developers and data scientists incorporate Spark into their applications to rapidly query, analyze, and transform data at scale. Tasks most frequently associated with Spark include ETL and SQL batch jobs across large data sets, processing of streaming data from sensors, IoT, or financial systems, and machine learning tasks.

**What is the need of spark?**

* High speed data querying, analysis, and transformation with large data sets.
* Compared to MapReduce, Spark offers much less reading and writing to and from the disk, multi-threaded tasks within Java Virtual Machine (JVM) processes
* Great for iterative algorithms (using a sequence of estimations based on the previous estimate).
* As mentioned earlier, the easy to use APIs make a big difference in terms of ease of development, readability, and maintenance.
* Super fast, especially for interactive queries. (100x faster than classic Hadoop Hive queries without refactoring the code!)
* Supports multiple languages and integrations with other popular products.
* Helps make complex data pipelines coherent and easy.

**2) Need of SparkContext.**

SparkContext is the entry point to any spark functionality. When we run any Spark application, a driver program starts, which has the main function and your SparkContext gets initiated here. The driver program then runs the operations inside the executors on worker nodes.

SparkContext uses Py4J to launch a **JVM** and creates a **JavaSparkContext**. By default, PySpark has SparkContext available as **‘sc’**, so creating a new SparkContext won't work.

A SparkContext represents **the connection to a Spark cluster**, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

Note: Only one SparkContext should be active per JVM.

**--Need of Sparksession:**

A [SparkContext](https://github.com/apache/spark/blob/master/core/src/main/scala/org/apache/spark/SparkContext.scala" \t "_blank) is a conduit to access all Spark functionality; only a single SparkContext exists per JVM. The Spark driver program uses it to connect to the cluster manager to communicate, submit Spark jobs and knows what resource manager (YARN, Mesos or Standalone) to communicate to. It allows you to configure Spark configuration parameters. And through SparkContext, the driver can access other contexts such as SQLContext, HiveContext, and StreamingContext to program Spark.

## 3).Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

**There are two ways to create RDD:**

**Parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

**Need of RDD:**

When it comes to iterative distributed computing, i.e. processing data over multiple jobs in computations such as  Logistic Regression, K-means clustering, Page rank algorithms, it is fairly common to reuse or share the data among multiple jobs or you may want to do multiple ad-hoc queries over a shared data set.

There is an underlying problem with data reuse or data sharing in existing distributed computing systems (such as MapReduce) and that is , you need to store data in some intermediate stable distributed store such as HDFS or Amazon S3. This makes the overall computations of jobs slower since it involves multiple IO operations, replications and serializations in the process.

**4).How to create RDD**

There are three ways to create an RDD in Spark.

* Parallelizing already existing collection in driver program.
* Referencing a dataset in an external storage system (e.g. HDFS, Hbase, shared file system).
* Creating RDD from already existing RDDs.

**5).Difference between Sequence and Lists in Scala.**

**Sequence in Scala**is a collection that stores elements in a fixed order. It is an indexed collection with 0 index.

**List is Scala**is a collection that stores elements in the form of a linked list.

**Declaration Syntax:**

Both have similar declaration syntax which is as follows:

Sequence : var coll\_Name = Seq(elementList)

list : var coll\_Name = List(elementList)

Both are collections that can store data but the sequence has some additional features over the list. In Scala, a list is a specialized collection that is optimized and commonly used in functional programming. There are some limitations of it but is commonly used to store data because of being optimized and faster compilation, also specialized functions available add a bit more functionality.