Hi. Welcome to the Spark Fundamentals course. This lesson will cover the Introduction to the Spark libraries.

Objectives:
After completing this lesson, you should be able to understand and use the various Spark libraries including SparkSQL, Spark Streaming, MLlib and GraphX

Slide 3:
Spark comes with libraries that you can utilize for specific use cases. These libraries are an extension of the Spark Core API. Any improvements made to the core will automatically take effect with these libraries. One of the big benefits of Spark is that there is little overhead to use these libraries with Spark as they are tightly integrated. The rest of this lesson will cover on a high level, each of these libraries and their capabilities. The main focus will be on Scala with specific callouts to Java or Python if there are major differences.

The four libraries are Spark SQL, Spark Streaming, MLlib, and GraphX.

Slide 4:

Spark SQL allows you to write relational queries that are expressed in either SQL, HiveQL, or Scala to be executed using Spark. Spark SQL has a new RDD called the SchemaRDD. The SchemaRDD consists of rows objects and a schema that describes the type of data in each column in the row. You can think of this as a table in a traditional relational database.

You create a SchemaRDD from existing RDDs, a Parquet file, a JSON dataset, or using HiveQL to query against the data stored in Hive.
You can write Spark SQL application using Scala, Java or Python.

Slide 5:
The SQLContext is created from the SparkContext. You can see here that in Scala, the sqlContext is created from the SparkContext. In Java, you create the JavaSQLContext from the JavaSparkContext. In Python, you also do the same.

There is a new RDD, called the SchemaRDD that you use with Spark SQL. In Scala only, you have to import a library to convert an existing RDD to a SchemaRDD. For the others, you do not need to import a library to work with the Schema RDD.

So how do these SchemaRDDs get created? There are two ways you can do this.

The first method uses reflection to infer the schema of the RDD. This leads to a more concise code and works well when you already know the schema while writing your Spark application. The second method uses a programmatic interface to construct a schema and then apply that to an existing RDD. This method gives you more control when you don't know the schema of the RDD until runtime. The next two slides will cover these two methods in more detail.

Slide 6:
The first of the two methods used to determine the schema of the RDD is to use reflection. In this scenario, use the case class in Scala to define the schema of the table. The arguments of the case class are read using reflection and becomes the names of the columns.

Let's go over the code shown on the slide. First thing is to create the RDD of the person object.

You load the text file in using the textFile method. Then you invoke the map transformation to split the elements on a comma to get the individual columns of name and age. The final transformation creates the Person object based on the elements.

Next you register the people RDD that you just created by loading in the text file and performing the transformation as a table.

Once the RDD is a table, you use the sql method provided by SQLContext to run SQL statements. The example here selects from the people table, the schemaRDD.

Finally, the results that comes out from the select statement is also a SchemaRDD.

That RDD, teenagers on our slide, can run normal RDD operations.

Slide 7:
The programmatic interface is used when cannot define the case classes ahead of time. For example, when the structure of records is encoded in a string or a text dataset will be parsed and fields will be projected different for different users.

A schemaRDD is created with three steps.

The first is to create an RDD of Rows from the original RDD. In the example, we create a schemaString of name and age.

The second step is to create the schema using the RDD from step one. The schema is represented by a StructType that takes the schemaString from the first step and splits it up into StructFields. In the example, the schema is created using the StructType by splitting the name and age schemaString and mapping it to the StructFields, name and age.

The third step convert records of the RDD of people to Row objects. Then you apply the schema to the row RDD.

Once you have your SchemaRDD, you register that RDD as a table and then you run sql statements against it using the sql method.

The results are returned as the SchemaRDD and you can run any normal RDD operation on it. In the example, we select the name from the people table, which is the SchemaRDD. Then we print it out on the console.