Hi. Welcome to the Spark Fundamentals course. This lesson will cover Resilient Distributed Dataset.

Objectives:
After completing this lesson, you should be able to understand and describe Spark's primary data abstraction, the RDD. You should know how to create parallelized collections from internal and external datasets. You should be able to use RDD opeartions such as Transformations and Actions. Finally, I will also show you how to take advantage of Spark's shared variables and key-value pairs.

Slide 3:
Resilient Distributed Dataset (RDD) is Spark's primary abstraction. RDD is a fault tolerant collection of elements that can be parallelized. In other words, they can be made to be operated on in parallel. They are also immutable. These are the fundamental primary units of data in Spark.

When RDDs are created, a direct acyclic graph (DAG) is created. This type of operation is called transformations. Transformations makes updates to that graph, but nothing actually happens until some action is called. Actions are another type of operations. We'll talk more about this shortly. The notion here is that the graphs can be replayed on nodes that need to get back to the state it was before it went offline – thus providing fault tolerance. The elements of the RDD can be operated on in parallel across the cluster. Remember, transformations return a pointer to the RDD created and actions return values that comes from the action.

There are three methods for creating a RDD. You can parallelize an existing collection. This means that the data already resides within Spark and can now be operated on in parallel. As an example, if you have an array of data, you can create a RDD out of it by calling the parallelized method. This method returns a pointer to the RDD. So this new distributed dataset can now be operated upon in parallel throughout the cluster.

The second method to create a RDD, is to reference a dataset. This dataset can come from any storage source supported by Hadoop such as HDFS, Cassandra, HBase, Amazon S3, etc.

The third method to create a RDD is from transforming an existing RDD to create a new RDD. In other words, let's say you have the array of data that you parallelized earlier. Now you want to filter out strings that are shorter than 20 characters. A new RDD is created using the filter method.

A final point on this slide. Spark supports text files, SequenceFiles and any other Hadoop InputFormat.

Slide 4:
Here is a quick example of how to create an RDD from an existing collection of data. In the examples throughout the course, unless otherwise indicated, we're going to be using Scala to show how Spark works. In the lab exercises, you will get to work with Python and Java as well. So the first thing is to launch the Spark shell. This command is located under the $SPARK_HOME/bin directory. In the lab environment, SPARK_HOME is the path to where Spark was installed.

Once the shell is up, create some data with values from 1 to 10,000. Then, create an RDD from that data using the parallelize method from the SparkContext, shown as sc on the slide. This means that the data can now be operated on in parallel.

We will cover more on the SparkContext, the sc object that is invoking the parallelized function, in our programming lesson, so for now, just know that when you initialize a shell, the SparkContext, sc, is initialized for you to use.

The parallelize method returns a pointer to the RDD. Remember, transformations operations such as parallelize, only returns a pointer to the RDD. It actually won't create that RDD until some action is invoked on it. With this new RDD, you can perform additional transformations or actions on it such as the filter transformation.

Another way to create a RDD is from an external dataset. In the example here, we are creating a RDD from a text file

using the textFile method of the SparkContext object. You will see plenty more examples of how to create RDD throughout this course.

Slide 5:
Here we go over some basic operations. You have seen how to load a file from an external dataset. This time, however, we are loading a file from the hdfs. Loading the file creates a RDD, which is only a pointer to the file. The dataset is not loaded into memory yet. Nothing will happen until some action is called. The transformation basically updates the direct acyclic graph (DAG).

So the transformation here is saying map each line s, to the length of that line. Then, the action operation is reducing it to get the total length of all the lines. When the action is called, Spark goes through the DAG and applies all the transformation up until that point, followed by the action and then a value is returned back to the caller.

A common example is a MapReduce word count. You first split up the file by words and then map each word into a key value pair with the word as the key, and the value of 1. Then you reduce by the key, which adds up all the value of the same key, effectively, counting the number of occurrences of that key. Finally, you call the collect() function, which is an action, to have it print out all the words and its occurrences.

Again, you will get to work with this in the lab exercises.