Hi. Welcome to the Spark Fundamentals course. This lesson will cover Spark application programming.

Objectives:
After completing this lesson, you should be able to understand the purpose and usage of the SparkContext.This lesson will show you how to can get started by programming your own Spark application. First you will see how to link to Spark. Next you will see how to run some Spark examples. You should also be able to understand how to pass functions to Spark and be able to create and run a Spark standalone application. Finally, you should be able to submit applications to the Spark cluster.

Slide 3:
The SparkContext is the main entry point to everything Spark. It can be used to create RDDs and shared variables on the cluster. When you start up the Spark Shell, the SparkContext is automatically initialized for you with the variable sc. For a Spark application, you must first import some classes and implicit conversions and then create the SparkContext object. The three import statements for Scala are shown on the slide here.

Slide 4:
Each Spark application you create requires certain dependencies. The next three slides will show you how to link to those dependencies depending on which programming language you decide to use.

To link with Spark using Scala, you must have a compatible version of Scala with the Spark you choose to use. For example, Spark 1.1.1 uses Scala 2.10, so make sure that you have Scala 2.10 if you wish to write applications for Spark 1.1.1.

To write a Spark application, you must add a Maven dependency on Spark. The information is shown on the slide here. If you wish to access a Hadoop cluster, you need to add a dependency to that as well.

In the lab environment, this is already set up for you. The information on this page shows how you would set up for your own Spark cluster.

Slide 5:
Spark 1.1.1 works with Python 2.6 or higher, but not Python 3. It uses the standard CPython interpreter, so C libraries like NumPy can be used.

To run Spark applications in Python, use the bin/spark-submit script located in the Spark's home directory. This script will load the Spark’s Java/Scala libraries and allow you to submit applications to a cluster. If you wish to use HDFS, you will have to link to it as well. In the lab environment, you will not need to do this as Spark is bundled with it. You also need to import some Spark classes shown here.

Slide 6:
Spark 1.1.1 works with Java 6 and higher. If you are using Java 8, Spark supports lambda expressions for concisely writing functions. Otherwise, you can use the org.apache.spark.api.java.function package with older Java versions.

As with Scala, you need to a dependency on Spark, which is available through Maven Central. If you wish to access an HDFS cluster, you must add the dependency there as well. Last, but not least, you need to import some Spark classes.

Slide 7:
Once you have the dependencies established, the first thing is to do in your Spark application before you can initialize Spark is to build a SparkConf object. This object contains information about your application.

For example, val conf = new SparkConf().setAppName(appName).setMaster(master).

You set the application name and tell it which is the master node. The master parameter can be a standalone Spark distribution, Mesos, or a YARN cluster URL. You can also decide to use the local keyword string to run it in local mode. In fact, you can run local[16] to specify the number of cores to allocate for that particular job or Spark shell as 16.

For production mode, you would not want to hardcode the master path in your program. Instead, launch it as an argument to the spark-submit command.

Once you have the SparkConf all set up, you pass it as a parameter to the SparkContext constructor to create the SparkContext

Slide 8:
Here's the information for Python. It is pretty much the same information as Scala. The syntax here is slightly different, otherwise, you are required to set up a SparkConf object to pass as a parameter to the SparkContext object. You are also recommended to pass the master parameter as an argument to the spark-submit operation.
.

Slide 9:
Here's the same information for Java. Same idea, you need to create the SparkConf object and pass that to the SparkContext, which in this case, would be a JavaSparkContext. Remember, when you imported statements in the program, you imported the JavaSparkContext libraries.