Slide 13:
In the next three slides, you will see another example of creating a Spark application. First you will see how to do this in Scala. The next following sets of slides will show Python and Java.

The application shown here counts the number of lines with ‘a’ and the number of lines with ‘b’. You will need to replace the YOUR_SPARK_HOME with the directory where Spark is installed, if you wish to code this application.

Unlike the Spark shell, you have to initialize the SparkContext in a program. First you must create a SparkConf to set up your application’s name. Then you create the SparkContext by passing in the SparkConf object. Next, you create the RDD by loading in the textFile, and then caching the RDD. Since we will be applying a couple of transformations on it, caching will help speed up the process, especially if the logData RDD is large. Finally, you get the values of the RDD by executing the count action on it. End the program by printing it out onto the console.

Slide 14:
In Python, this application does the exact same thing, that is, count the number of lines with ‘a’ in it, and the number of lines with ‘b’ in it. You use a SparkContext object to create the RDDs and cache it. Then you run the transformations and actions, follow by a print to the console. Nothing entirely new here, just a difference in syntax.

Slide 15:
Similar to Scala and Python, in Java you need to get a JavaSparkContext. RDDs are represented by JavaRDD. Then you run the transformations and actions on them. The lambda expressions of Java 8 allows you to concisely write functions. Otherwise, you can use the classes in the org.apache.spark.api.java.function package for older versions of java. The business logic is the same as the previous two examples to count the number of a's and b's from the Readme file. Just a matter of difference in the syntax and library names.

Slide 16:
Up until this point, you should know how to create a Spark application using any of the supported programming languages. Now you get to see how to run the application. You will need to first define the dependencies. Then you have to package the application together using system build tools such as Ant, sbt, or Maven. The examples here show how you would do it using the various tools. You can use any tool for any of the programming languages. For Scala, the example is shown using sbt, so you would have a simple.sbt file. In Java, the example shows using Maven so you would have the pom.xml file. In Python, if you need to have dependencies that requires third party libraries, then you can use the –py-files argument to handle that.

Again, shown here are examples of what a typical directory structure would look like for the tool that you choose.

Finally, once you have the JAR packaged created, run the spark-submit to execute the application.

In the lab exercise, you will get to practice this.

Slide 17:
In short, you package up your application into a JAR for Scala or Java or a set of .py or .zip files for Python.

To submit your application to the Spark cluster, you use spark-submit command, which is located under the $SPARK_HOME/bin directory.

The options shown on the slide are the commonly used options. To see other options, just invoke spark-submit with the help argument.

Let's briefly go over what each of these options mean.

The class option is the main entry point to your class. If it is under a package name, you must provide the fully qualified name. The master URL is where your cluster is located. Remember that it is recommended approach to provide the master URL here, instead of hardcoding it in your application code.

The deploy-mode is whether you want to deploy your driver on the worker nodes (cluster) or locally as an external client (client). The default deploy-mode is client.

The conf option is any configuration property you wish to set in key=value format.

The application jar is the file that you packaged up using one of the build tools.

Finally, if the application has any arguments, you would supply it after the jar file.

Here's an actual example of running a Spark application locally on 8 cores. The class is the org.apache.spark.examples.SparkPi. local[8] is saying to run it locally on 8 cores. The examples.jar is located on the given path with the argument 100 to be passed into the SparkPi application.

Slide 18
Having completed this lesson, you should now know how to create a standalone Spark application and run it by submitting it to the Spark Cluster. You saw briefly on the different methods on how to pass functions in Spark. All three programming languages were shown in this lesson.

Slide 19
Next steps. Copmlete lab exercise #3, Creating a Spark application and then proceed to the next lesson in this course.