Slide 8:
Spark runs on Windows or Unix-like systems. The lab exercises that are part of this course will be running with IBM BigInsights. The information on this slide shows you how you can install the standalone version yourself with all the default values that come along with it.

To install Spark, simply download the pre-built version of Spark and place a compiled version of Spark on each node of your cluster. Then you can manually start the cluster by executing ./sbin/start-master.sh . In the lab, the start script is different, and you will see this in the lab exercise guide.

Again, the default URL to the master UI is on port 8080. The lab exercise environment will be different and you will see this in the guide as well.

You can check out Spark's website for more information:
http://spark.apache.org/docs/latest/spark-standalone.html

Slide 9:
Spark jobs can be written in Scala, Python or Java. Spark shells are available for Scala or Python. This course will not teach how to program in each specific language, but will cover how to use them within the context of Spark. It is recommended that you have at least some programming background to understand how to code in any of these.

If you are setting up the Spark cluster yourself, you will have to make sure that you have a compatible version of the programming language you choose to use. This information can be found on Spark's website. In the lab environment, everything has been set up for you – all you do is launch up the shell and you are ready to go.

Spark itself is written in the Scala language, so it is natural to use Scala to write Spark applications. This course will cover code examples from by Scala, Python, and Java. Java 8 actually supports the functional programming style to include lambdas, which concisely captures the functionality that are executed by the Spark engine. This bridges the gap between Java and Scala for developing applications on Spark. Java 6 and 7 is supported, but would require more work and an additional library to get the same amount of functionality as you would using Scala or Python.

Slide 10:
Here we have a brief overview of Scala. Everything in Scala is an object. The primitive types that is defined by Java such as int or boolean are objects in Scala. Functions are objects in Scala and will play an important role in how applications are written for Spark.

Numbers are objects. This means that in an expression that you see here: 1 + 2 * 3 / 4 actually means that the individual numbers invoke the various identifiers +,-,*,/ with the other numbers passed in as arguments using the dot notation.

Functions are objects. You can pass functions as arguments into another function. You can store them as variables. You can return them from other functions. The function declaration is the function name followed by the list of parameters and then the return type.

This slide and the next is just to serve as a very brief overview of Scala. If you wish to learn more about Scala, check out its website for tutorials and guide. Throughout this course, you will see examples in Scala that will have explanations on what it does. Remember, the focus of this course is on the context of Spark. It is not intended to teach Scala, Python or Java.

Slide 11:
Anonymous functions are very common in Spark applications. Essentially, if the function you need is only going to be required once, there is really no value in naming it. Just use it anonymously on the go and forget about it. For example, suppose you have a timeFlies function and it in, you just print a statement to the console. In another function, oncePerSecond, you need to call this timeFlies function. Without anonymous functions, you would code it like the top

example be defining the timeFlies function. Using the anonymous function capability, you just provide the function with arguments, the right arrow, and the body of the function after the right arrow as in the bottom example. Because this is the only place you will be using this function, you do not need to name the function.

Slide 12:
The Spark shell provides a simple way to learn Spark's API. It is also a powerful tool to analyze data interactively. The Shell is available in either Scala, which runs on the Java VM, or Python. To start up Scala, execute the command spark-shell from within the Spark's bin directory. To create a RDD from a text file, invoke the textFile method with the sc object, which is the SparkContext. We'll talk more about these functions in a later lesson.

To start up the shell for Python, you would execute the pyspark command from the same bin directory. Then, invoking the textFile command will also create a RDD for that text file.

In the lab exercise, you will start up either of the shells and run a series of RDD transformations and actions to get a feel of how to work with Spark. In a later lesson and exercise, you will get to dive deeper into RDDs.

Slide 13:
Having completed this lesson, you should now understand what Spark is all about and why you would want to use it. You should be able to list and describe the components in the Spark stack as well as understand the basics of Spark's primary abstraction, the RDDs. You also saw how to download and install Spark's standalone if you wish to, or you can use the provided lab environment. You got a brief overview of Scala and saw how to launch and use the two Spark Shells.

Slide 14:
You have completed this lesson. Go on to the first lab exercise and then proceed to the next lesson in this course.