**Big Data Analysis**

**Introduction to Data Analysis with Scala and Spark**

Ji-Hyeong Han

## ( [jhhan@seoultech.ac.kr](mailto:jhhan@seoultech.ac.kr) )

Dept. of Computer Science and Engineering



**Today’s Contents**

1. Prepare the Docker & Jupyter Notebook environment
2. Scala for data scientists
3. The Spark programming model
4. Getting Started: Spark Shell and SparkContext
5. Analyzing the real data using Scala and Spark
6. More Practice

2

# Prepare Docker & Jupyter Notebook Environment

3

**Install Docker**

## Download “Docker Desktop” based on your OS

* + <https://www.docker.com/get-started>

## Install Docker

* Please refer the supplement material to install docker in e-class

4

**Jupyter Notebook**

* We’re going to use Jupyter Notebook to practice Scala and Spark
  + Please refer <https://github.com/SeoulTech-HCIRLab/spark_notebook>

1. Run “cmd” as Administrator
2. mkdir <your workspace directory>
3. docker run -h "localhost" -v <your workspace directory>:/root/workspace -p 8888:8888 -p 8080:8080 -p 8088:8088 -p 4040:4040 -p 50070:50070 titania7777/spark2\_notebook:1

example) docker run -h "localhost" -v D:/docker-workspace/Spark:/root/workspace -p 8888:8888 -p 8080:8080 -p 8088:8088 -p 4040:4040 -p 50070:50070

titania7777/spark2\_notebook:1

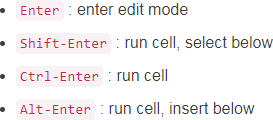
1. Connect the URL (http://localhost:8888) using web browser (chrome, IE, etc)
2. At the Jupyter Notebook, click “New”, choose “spylon-kernel”

* You can view all Docker containers
  + Right button click on Docker icon and select “Dashboard”
  + You can start and stop each container
* If you have a trouble to follow above steps, then please contact TA

5

**Jupyter Notebook**

* Keyboard shortcuts of Jupyter Notebook [https://www.cheatography.com/weidadeyue/cheat-sheets/jupyter- notebook/](https://www.cheatography.com/weidadeyue/cheat-sheets/jupyter-notebook/)



6

# Scala for data scientists

7

**Scala for data scientists**

## Other favorite tool for data analyzation

* + R, Python

## Spark also has Python APIs, which is PySpark

* But, learning how to work with Spark in the same language (i.e. Scala) in which the underlying framework is written has a number of advantages

8

**Scala for data scientists**

## Advantages of using Scala in Spark

* + Reduce performance overhead
    - When we’re running an algorithm in R or Python,

we have to do some work to pass code and data across the different environments

→ Sometimes, things can get lost in this translation

* + - When we’re running an algorithm in Spark with Scala API,

we can be far more confident that our program will run as intended

* + Access to the latest and greatest
    - All of Spark’s machine learning, stream processing, and graph analytics libraries are written in Scala
    - Python and R bindings tend to get support this new functionality much later
  + Help you understand the Spark philosophy
    - Python or R APIs reflect the underlying computation philosophy of Scala
    - If you know how to use Spark in Scala,

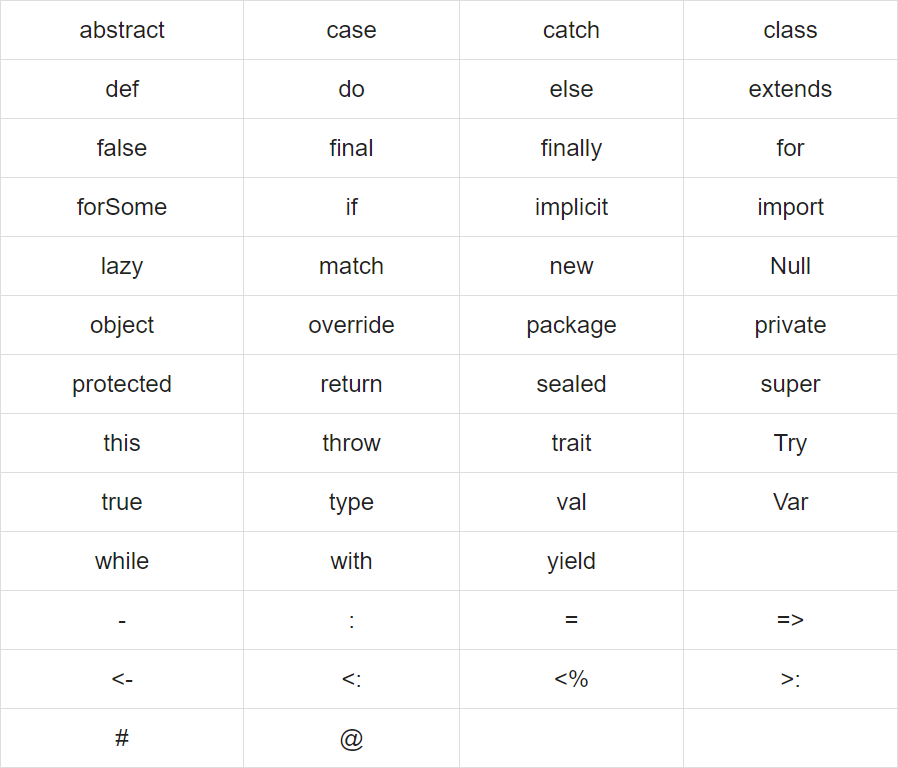
you’ll have a better understanding of the system

9

**Scala Tutorials**

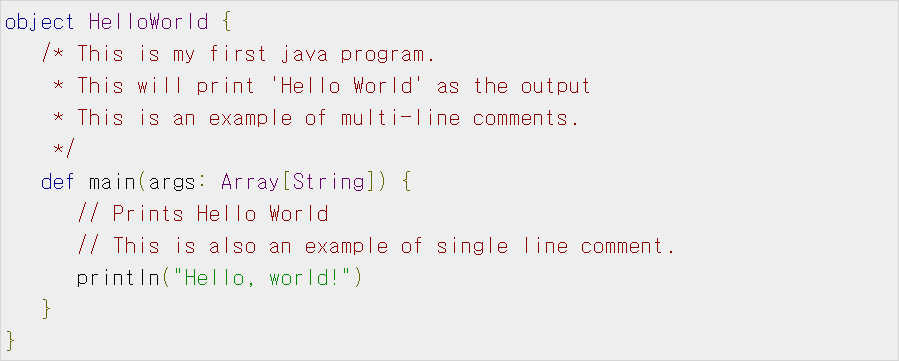
* <https://www.tutorialspoint.com/scala/index.htm>
* Identifiers
  + Legal: age, salary, \_value, 1\_value
  + Ilegal: $salary, 123abc, -salary

## Keywords

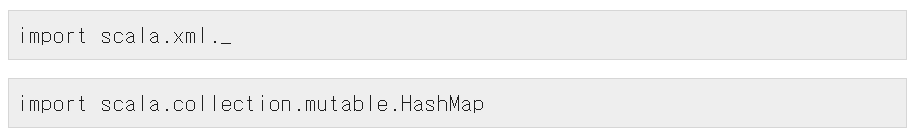
 10

**Scala Tutorials**

## Comments



* Import packages





11

**Scala Tutorials**

## Data Types

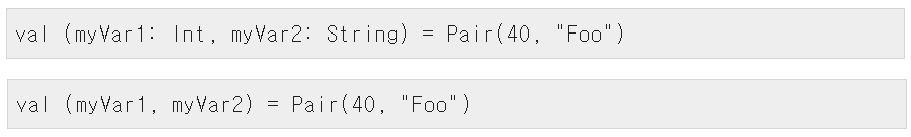
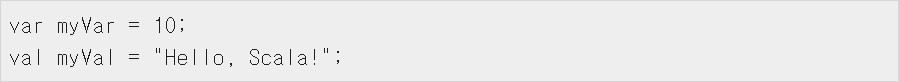
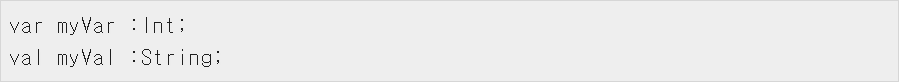
|  |
| --- |
| **Byte** 8 bit signed value. Range from -128 to 127 |
| **Short** 16 bit signed value. Range -32768 to 32767 |
| **Int** 32 bit signed value. Range -2147483648 to 2147483647 |
| **Long** 64 bit signed value. -9223372036854775808 to 9223372036854775807 |
| **Float** 32 bit IEEE 754 single-precision float |
| **Double** 64 bit IEEE 754 double-precision float |
| **Char** 16 bit unsigned Unicode character. Range from U+0000 to U+FFFF |
| **String** A sequence of Chars |
| **Boolean** Either the literal true or the literal false |
| **Unit** Corresponds to no value |
| **Null** null or empty reference |
| **Nothing** The subtype of every other type; includes no values |
| **Any** The supertype of any type; any object is of type *Any* |
| **AnyRef** The supertype of any reference type |

12

**Scala Tutorials**

## Variable

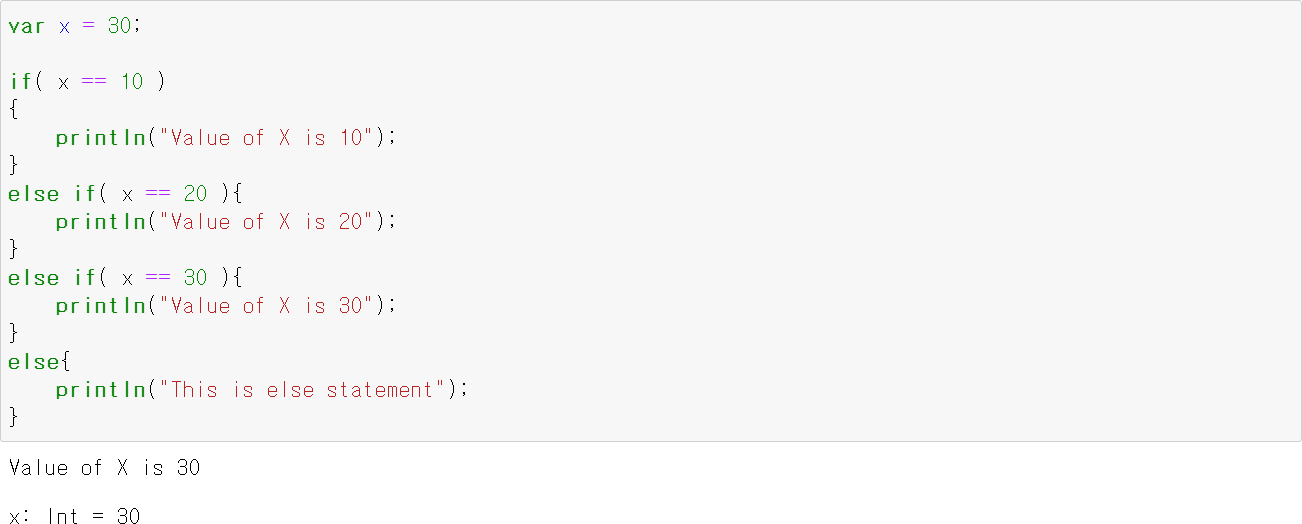
* + var: variable (mutable)
  + val: constant (immutable)



13

**Scala Tutorials**

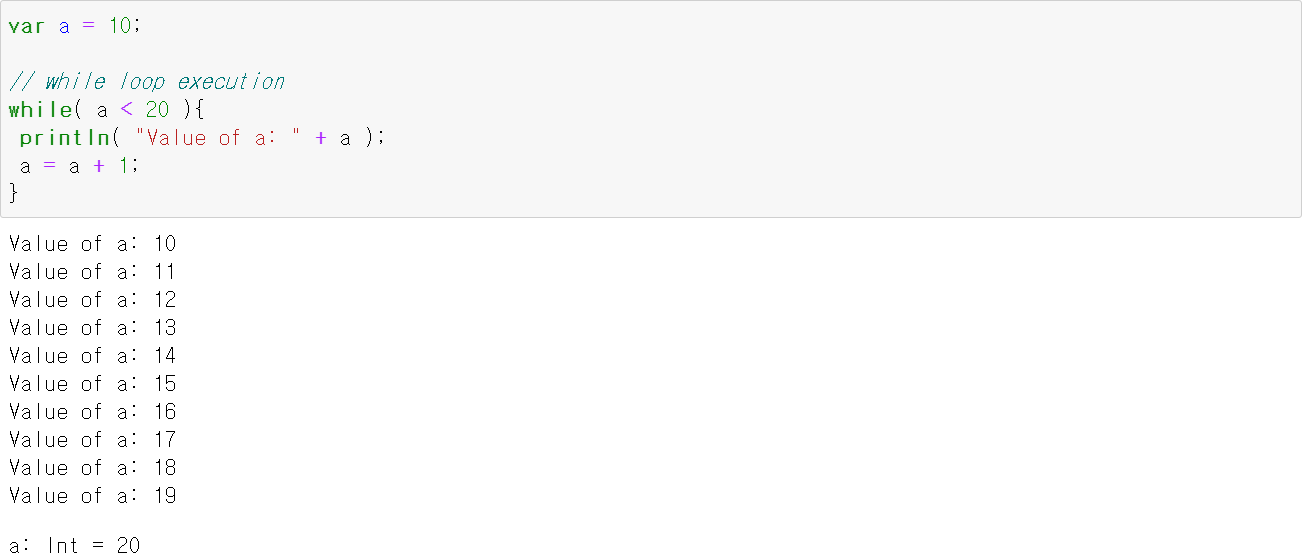
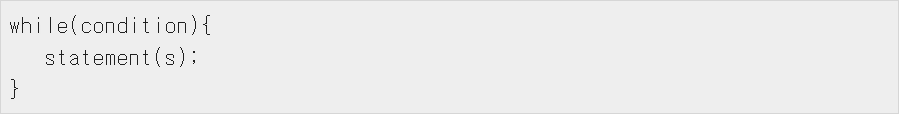
## if-else-elseif



14

**Scala Tutorials**

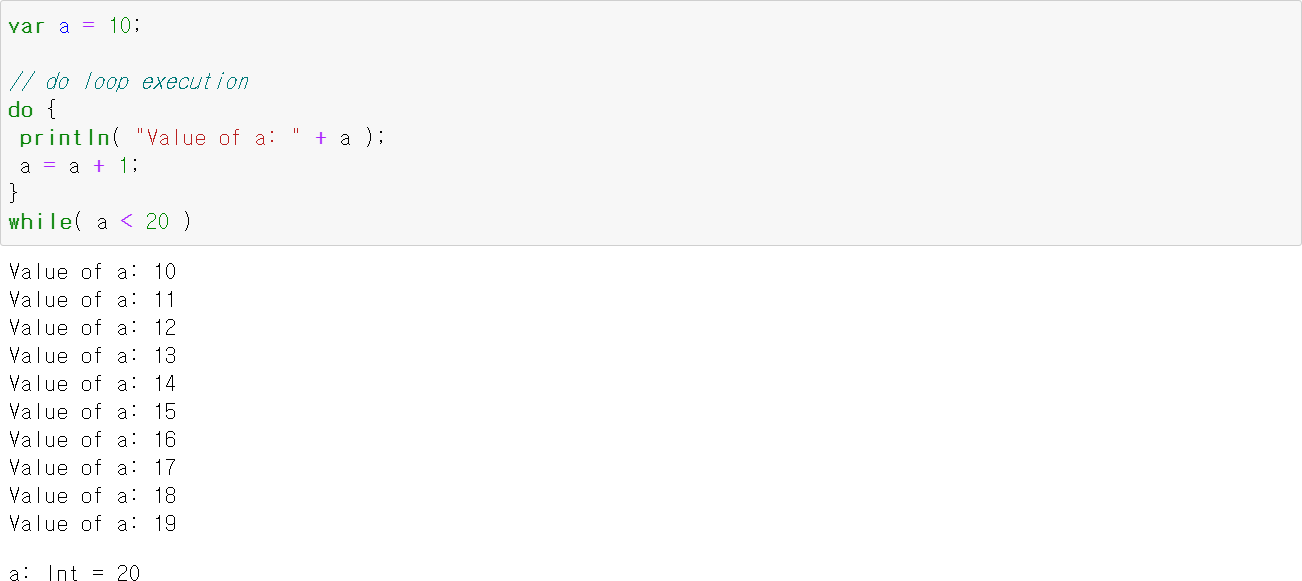
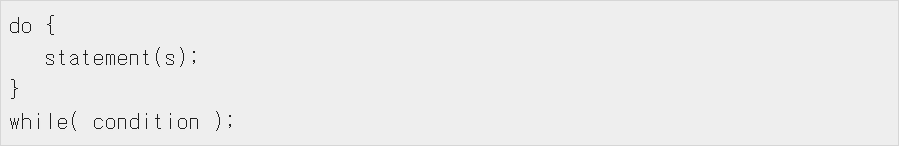
## while



15

**Scala Tutorials**

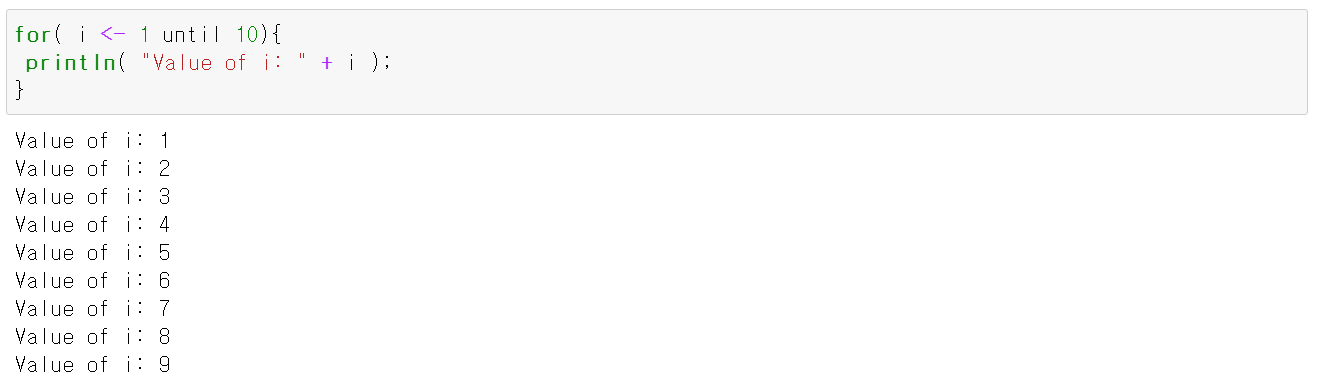
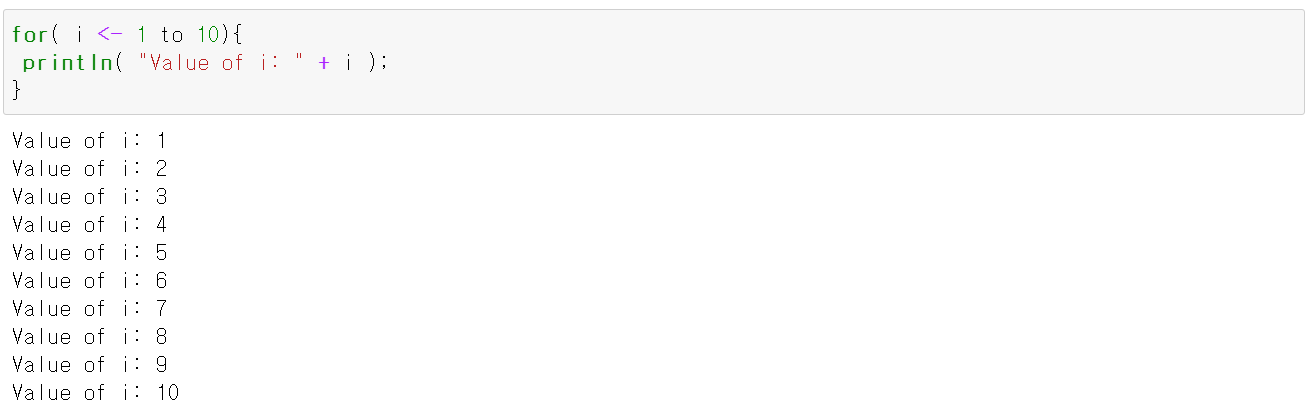
## do-while



16

**Scala Tutorials**

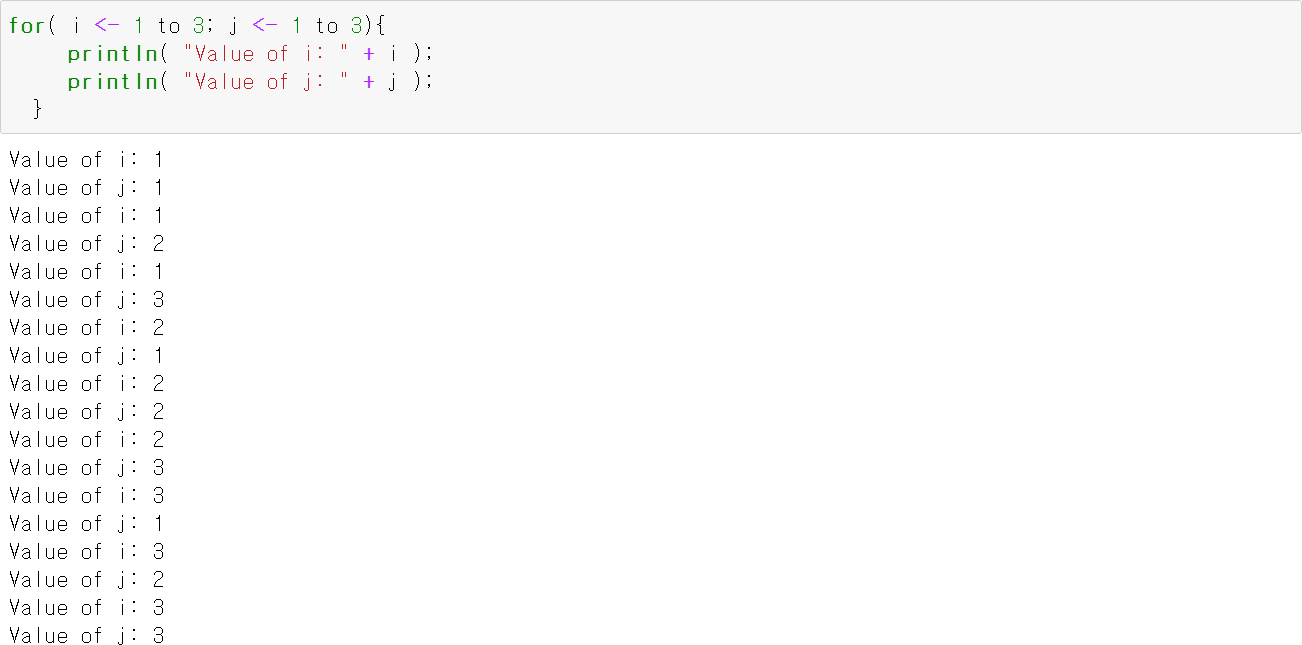
## for loop with ranges



17

**Scala Tutorials**

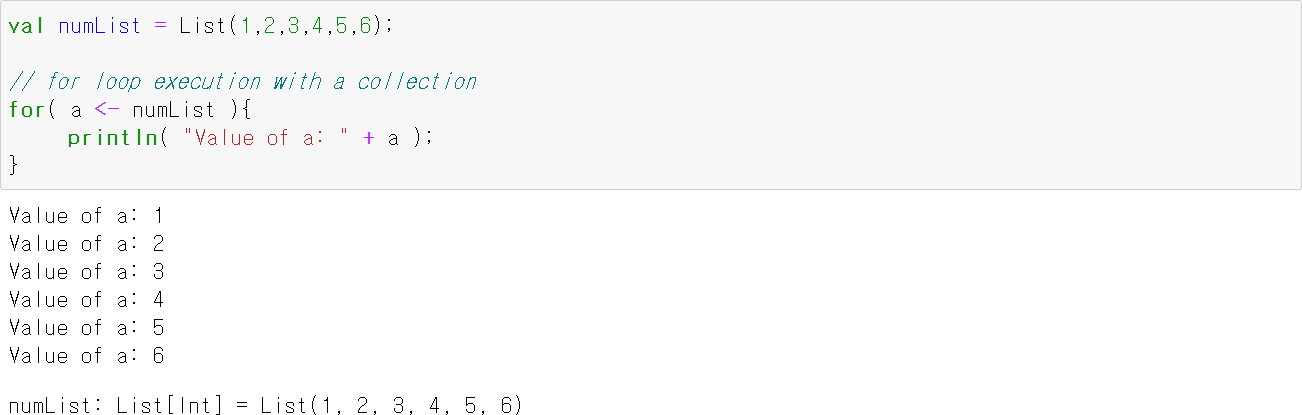
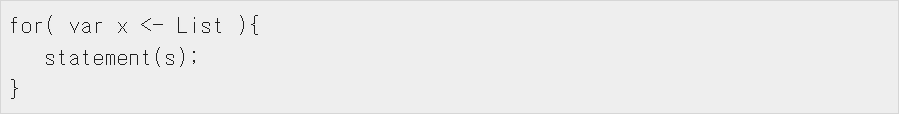
## for loop with ranges



18

**Scala Tutorials**

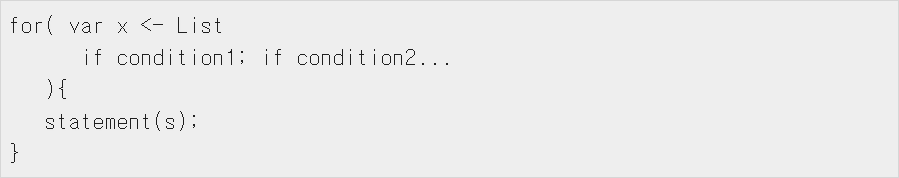
## for loop with collections

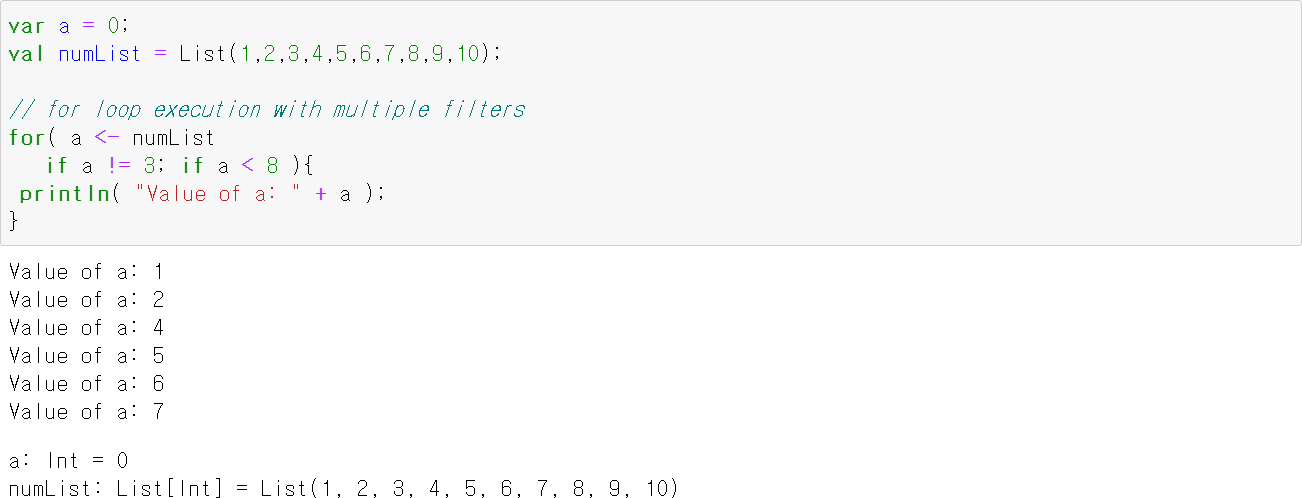


19

**Scala Tutorials**

## for loop with filters

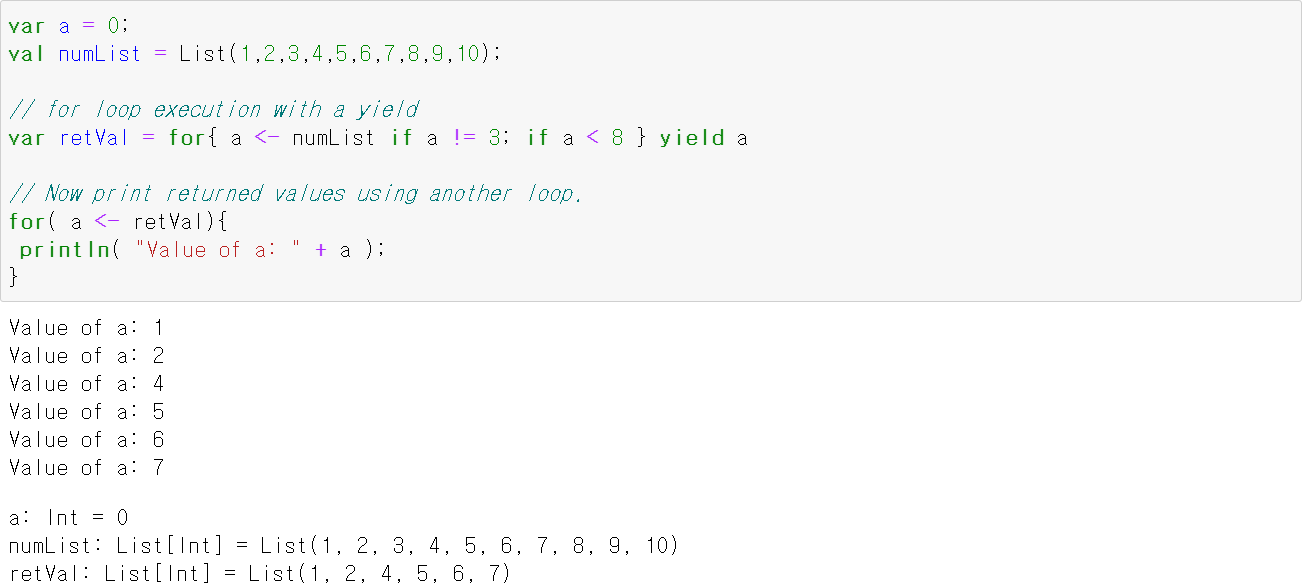
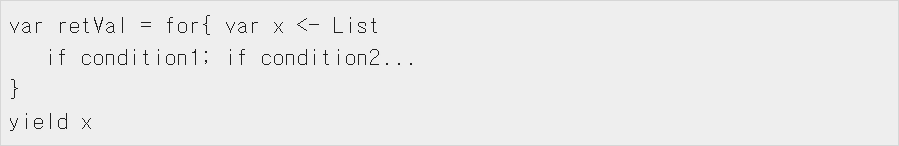




20

**Scala Tutorials**

## for loop with yield



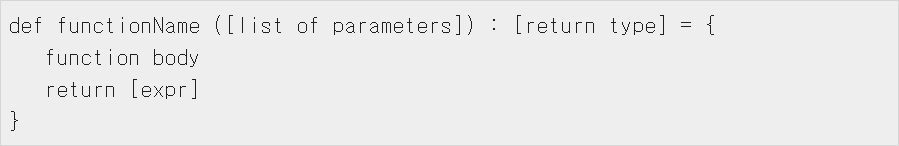
21

**Scala Tutorials**

## Function declarations



* Function definitions



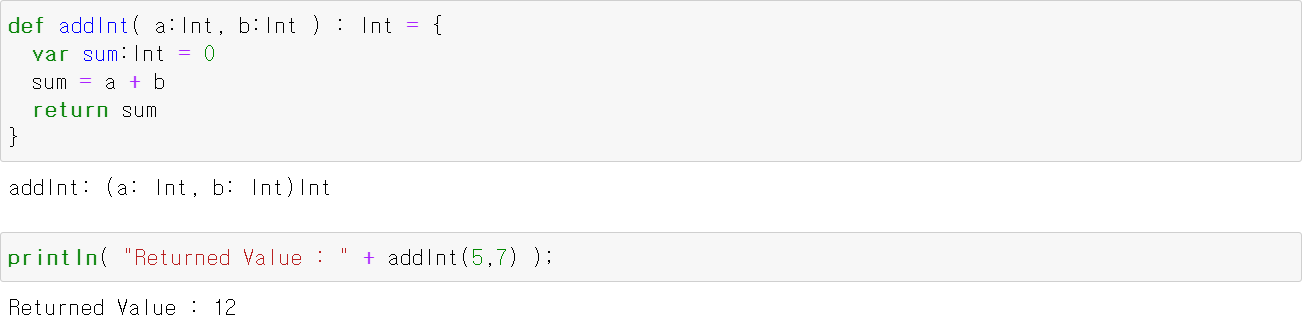
* Function calling



22

**Scala Tutorials**

## Function example



23

**Scala Tutorials**

## Anonymous function

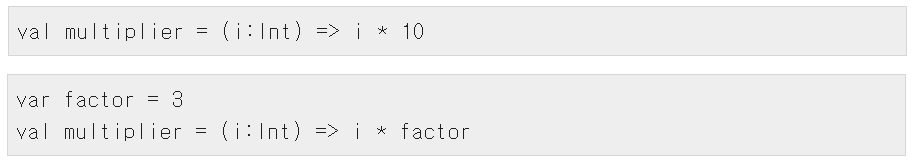


24

**Scala Tutorials**

## closure

* + A function, whose return value depends on the value of one or more variables declared outside this function
  + Anonymous function



25

**Scala Tutorials**

## Collections

|  |
| --- |
| [Scala Lists](https://www.tutorialspoint.com/scala/scala_lists.htm)  Scala's List[T] is a linked list of type T. |
| [Scala Sets](https://www.tutorialspoint.com/scala/scala_sets.htm)  A set is a collection of pairwise different elements of the same type. |
| [Scala Maps](https://www.tutorialspoint.com/scala/scala_maps.htm)  A Map is a collection of key/value pairs. Any value can be retrieved based on its key. |
| [Scala Tuples](https://www.tutorialspoint.com/scala/scala_tuples.htm)  Unlike an array or list, a tuple can hold objects with different types. |
| [Scala Options](https://www.tutorialspoint.com/scala/scala_options.htm)  Option[T] provides a container for zero or one element of a given type. |
| [Scala Iterators](https://www.tutorialspoint.com/scala/scala_iterators.htm)  An iterator is not a collection, but rather a way to access the elements of a collection one by one. |

26

**Scala Tutorials**

## Class & Object

* + Once you define a class, you can create objects from the class blueprint with the keyword **new**

## Class

27

# Spark Programming Model

28

**Spark Programming Model**

## Writing a Spark program typically consists of a few related steps

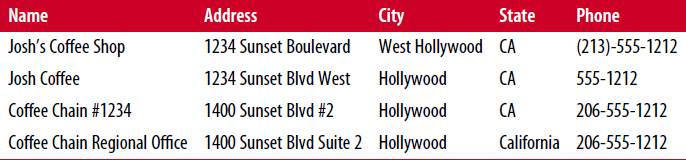
* + 1. Define a set of transformations on the input data set
    2. Invoke actions that output the transformed data sets to persistent storage or return results to the driver’s local memory
    3. Run local computations that operate on the results computed in a distributed fashion

29

**Record Linkage**

## We have a large collection or records from one or more source systems

* It is likely that multiple records refer to the same underlying entity
* Example of record linkage challenge



30

# Getting Started: Spark Shell and

**SparkContext**

31

**Download Data**

## We’re going to use a sample data set from the UC Irvine Machine Learning Repository

* Data
  + Record linkage study performed at a German hospital in 2010
  + Contain several million pairs of patient records (patient’s first and last name, address, birthday)
  + Each matching field was assigned a numerical score from 0.0 to 1.0 based on how similar the strings were
  + The data was hand-labeled to identify which pairs represented the same person and which did not
  + The underlying values of the fields that were used to create the data set were removed to protect the privacy of the patients
  + Numerical identifiers, the match scores for the fields, and the label for each pair (match versus nonmatch) were published for use in record linkage research

32

**Download Data**

## In jupyter notebook, open terminal

* + new -> terminal

# apt-get install curl unzip # mkdir Data

# cd Data

# mkdir linkage # cd linkage

# curl –L –o donation.zip <https://bit.ly/1Aoywaq> # unzip donation.zip

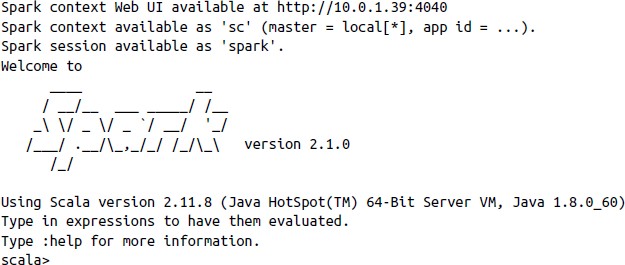
# unzip ‘block\_\*.zip’

# rm block\_\*.zip documentation donation.zip frequencies.csv

33

**Spark Shell & SparkContext**

## If we run the Spark Shell at the own system, we can see this Spark Shell



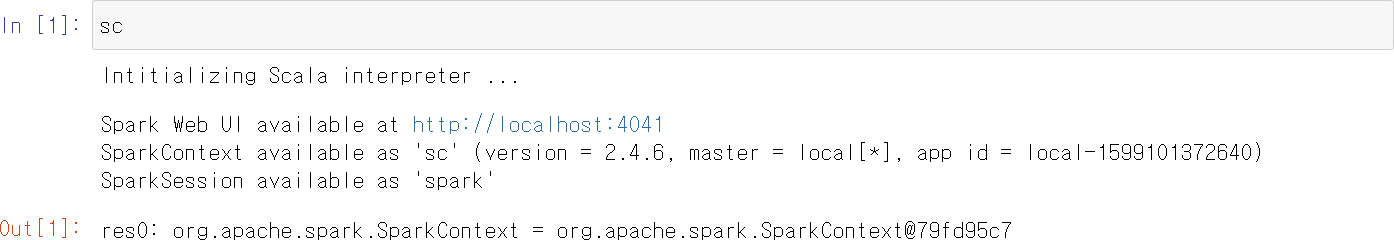
34

**Spark Shell & SparkContext**

## SparkContext (sc)

* + sc is an object & sc has methods
  + We’re going to use most often to create

*Resilient Distributed Datasets* (*RDDs*)



35

**RDD (Resilient Distributed Datasets)**

## Spark’s fundamental abstraction for representing a collection of objects that can be distributed across multiple machines in a cluster

* Two ways to create an RDD in Spark
  + Using the *SparkContext* to create an RDD from an external data source
  + Performing a transformation on one or more existing RDDs

(ex: filtering records, aggregating records by a common key, joining multiple RDDs together)

36

**RDD Exercise**

## The simplest way to create an RDD is to use *parallelize* method on

*SparkContext* with a local collection of objects



collection of objects to parallelize number of partitions

## To create an RDD from a text file or directory



* + When we’re running Spark in local mode, we can use *textFile* method to access the local filesystem
  + If Spark is given a directory instead of an individual file,

it will consider all of the files in that directory as part of the given RDD

37

# Analyzing the real data using Scala and Spark

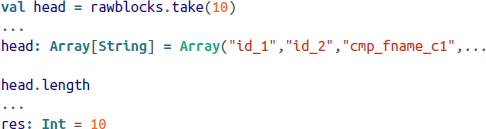
38

**Bringing Data from the Cluster to the Client**

* *first* method



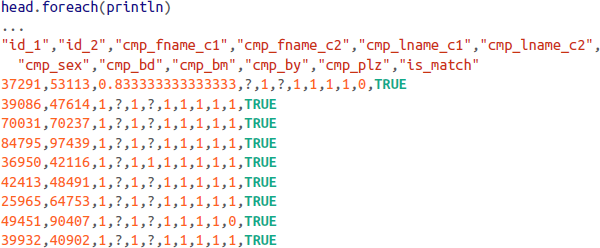
* *take* method



39

**Bringing Data from the Cluster to the Client**

* *foreach* method



* + To make the raw data easier to read the contents of an array
  + We can pass one function (*println*) as an argument to another function (*foreach*)

40

**Bringing Data from the Cluster to the Client**

## CSV files contain a header row that we’ll want to filter out from our subsequent analysis

* In Scala, we declare functions using the keyword *def*
* We have to specify the types of arguments to our function In this case, *line* argument is a *String*
* The body of the function, using the *contains* method for the String class to test whether or not the characters “id\_1” appear anywhere in the string

## Test our new Scala function against the data in the *head* array by using the *filter* method



41

**Bringing Data from the Cluster to the Client**

## But, we want to get all of the rows in the data except the header rows



42

**Shipping Code from the Client to the Cluster**

## Until now, all the code that we executed was done against the data inside the *head* array

And it was contained on our client machine

* Now, we’re going to take the code and apply it to the millions of linkage records contained in our cluster and represented by *rawblocks* RDD in Spark





43

**From RDDs to Data Frames**

## DataFrame

* + An abstraction built on top of RDDs for data sets that have a regular structure
  + Each record is a row made up of a set of columns
  + Each column has a well-defined data type
  + You can think of a DataFrame as the Spark analogue of a table in a relational database
  + Can represent distributed data sets on a cluster,

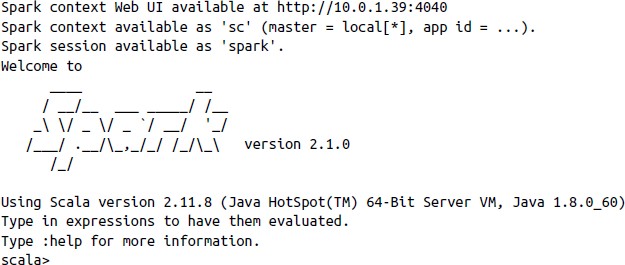
not local data where every row in the data is stored on the same machine

44

**From RDDs to Data Frames**

## To create a DataFrame for our record linkage data set,

we’re going to use the other object, *spark* instead of *sparkcontext*



45

**From RDDs to Data Frames**

## Before create DataFrames, first we need to delete the all zip files in the directory

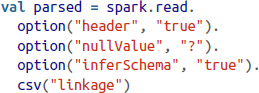
46

**From RDDs to Data Frames**

## Create DataFrames for our data



* We can read and parse the linkage data like this

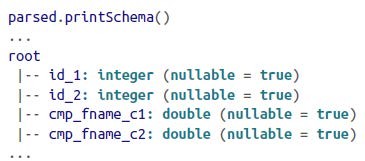


* + To deal with “?” strings in some of the columns
  + To handle these as missing values
  + To name each column correctly
  + To infer the data type of each of the columns

47

**From RDDs to Data Frames**

## Now we have



48

**Analyzing Data with the DataFrame API**

## If we look at the schema of the parsed DataFrame and the first few rows of data, we see this:

* + The first two fields are integer IDs that represent the patients that were matched in the record.
  + The next nine values are (possibly missing) numeric values (either doubles or ints) that represent match scores on different fields of the patient records, such as their names, birthdays, and locations. The fields are stored as integers when the only possible values are match (1) or no- match (0), and doubles whenever partial matches are possible.
  + The last field is a boolean value (true or false) indicating whether or not the pair of patient records represented by the line was a match.

## Our goal is to come up with a simple classifier that allows us to predict whether a record will be a match based on the values of the match scores for the patient records.

49

**Analyzing Data with the DataFrame API**

## Let’s start by getting an idea of the number of records we’re dealing with via the count method



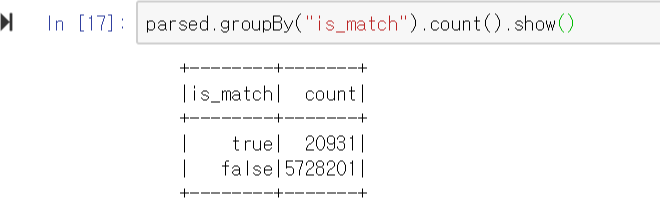
* After the data has been parsed once, we’d like to save the data in its parsed form on the cluster so that we don’t have to reparse it every time we want to ask a new question of the data.
  + Given RDD or DataFrame should be cached in memory after it is generated by calling the cache method on the instance.



50

**Analyzing Data with the DataFrame API**

## Once our data has been cached, the next thing we want to know is the relative fraction of records that were matches versus those that were nonmatches.

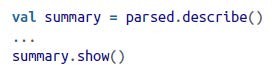


* Two ways we can reference the names of the columns
  + Literal string, like in groupBy(“is\_match”)
  + Column objects by using the special $”<col>”

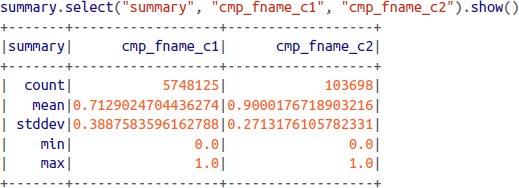
51

**Fast Summary Statistics for DataFrames**

## Fast summary statistics



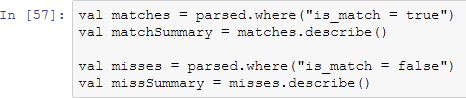
* We can use the select method to choose a subset of the columns in order to make the summary statistics easier to read and compare:



52

**Fast Summary Statistics for DataFrames**

## Our next step is to compute those same summary statistics for just the subsets of the parsed DataFrame that correspond to matches and nonmatches.

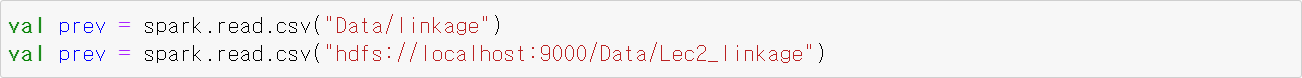
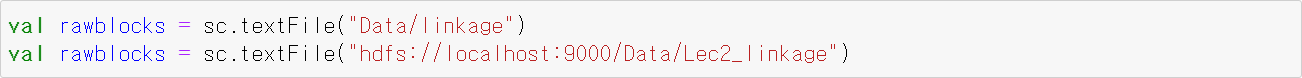


53

**Using data from Hadoop**

## In our docker image, Hadoop is already installed and practice datasets are uploaded in HDFS

* + If you want to use the data through Hadoop, you can use HDFS path instead of local file directory as follows



54

# More Practice

55

**Practice #1 - Scala**

## There are more tutorials !

* + <https://www.tutorialspoint.com/scala/>

56

**Practice #2 – Data Analyzation**

## Download movielens 100kb data sets by using terminal

* + wget <http://files.grouplens.org/datasets/movielens/ml-100k.zip>
  + unzip the data sets
  + important file: README (information about data sets) u.user (user profile)

u.item (movie metadata) u.data (movie score by users)

57

**Practice #2 – Data Analyzation**

## Data analyzation

1. Load each data file as RDD
2. Take a look of each data file
3. Print top-10 data from each data file
4. Load each data file as DataFrame

- Reference: <https://github.com/databricks/spark-csv#features>

## Print schema of each data file

1. Make fast summary of each data file
2. Calculate average, std, min, max score of each movie
3. Calculate average, std, min, max score of each user

58