

# 빅데이터 분석을 위한 Spark



# What is Spark?

**A fast and general engine for large-scale data processing**

**It's scalable**

**It's fast**

**It's hot**

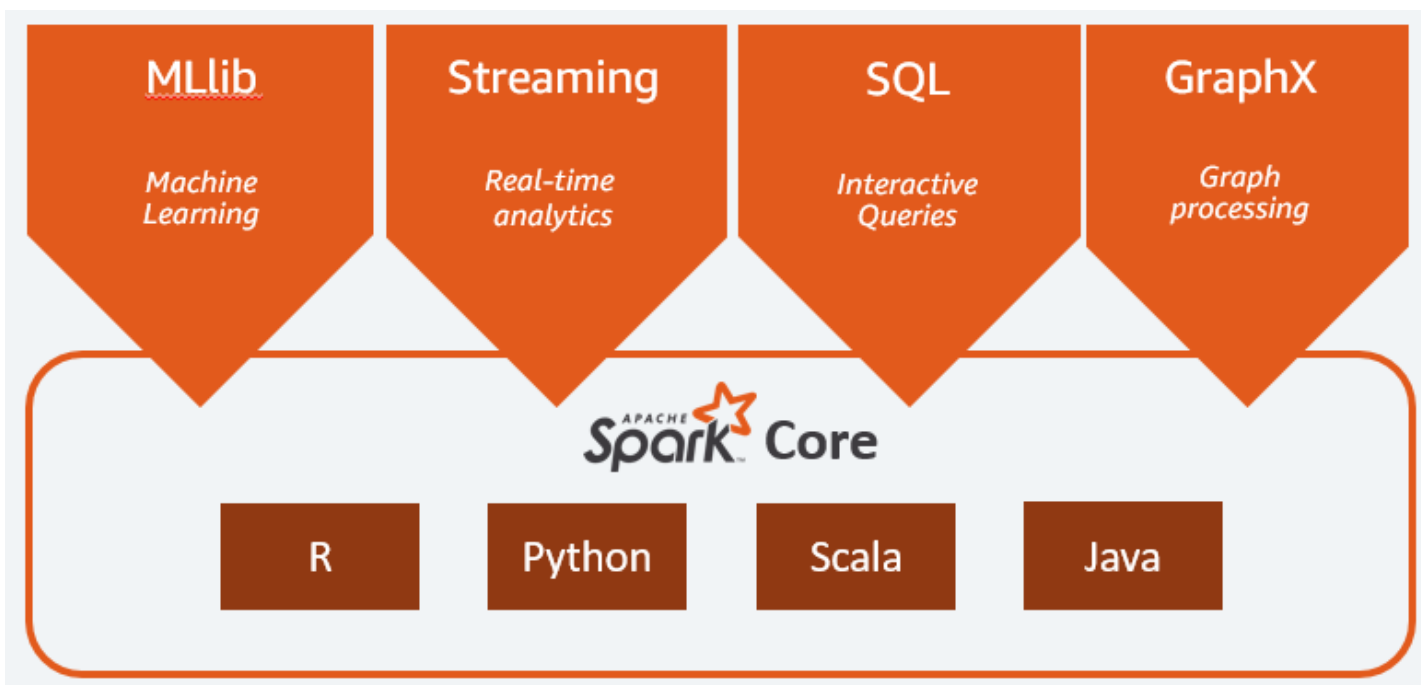
**It's not that hard**

Code in Python, Java, or Scala

Built around on main concept : RDD(Resilient Distributed Dataset)




# What is Spark?

- 인메모리 기반의 대용량 데이터 고속 처리 엔진으로 범용 분산 클러스터 컴퓨팅 프레임워크입니다.
- UC Berkeley의 AMPLab에서 만들었으며, 사용 편의성과 속도로 인기가 높습니다.
- Spark를 MapReduce의 flexible alternative 으로 생각할 수 있습니다.
- Spark는 Cassandra, AWS S3, HDFS 등 다양한 형식으로 저장된 데이터를 사용할 수 있습니다.
- Apache Spark는 구조화된 데이터를 처리하기 위한 **Spark SQL**과 머신러닝을 위한 **ML Lib**, 그래프 처리를 위한 **Graph X**, 실시간 처리와 방대한 연산을 위한 **Structured Streaming** 도구를 제공합니다.
- Scala 언어로 만들어졌지만 Java, R, Python 등과 같은 다양한 프로그래밍 언어 API를 제공한다.



# Spark vs MapReduce

- MapReduce는 각 map, reduce 이후에 대부분의 데이터를 디스크(HDFS)에 저장합니다.
- Spark는 각 변환 후 대부분의 데이터를 메모리에 보관합니다 .
- Spark는 인메모리(In-Memory) 기반의 처리로 하둡의 맵리듀스에 비해서 100배 빠른 속도를 제공하고, 머신러닝, 그래프처리 등 빅데이터 분석을 위한 통합 컴포넌트를 제공합니다.

Spark  vs  Hadoop MapReduce		
Factors	Spark 	Hadoop MapReduce
Speed	100x times than MapReduce	Faster than traditional system
Written In	Scala	Java
Data Processing	Batch / real-time / iterative / interactive / graph	Batch processing
Ease of Use	Compact & easier than Hadoop	Complex & lengthy
Caching	Caches the data in-memory & enhances the system performance	Doesn't support caching of data

# 버전별 특징

## ■ Spark V1

- RDD를 이용한 인메모리 처리로 기존 방식 대비 빠른 속도로 처리가 가능함
- V1.3에서 데이터프레임 추가 : 프로젝트 텅스텐
- V1.6에서 데이터셋 추가 : 데이터 타입체크, 인코더, 카탈리스트 옵티마이저 지원

## ■ Spark V2

- 2016년 RDD의 기능을 개선한 데이터프레임과 데이터셋을 통합한 V2가 발표 되었습니다.
- 데이터를 스키마 형태로 추상화 하여 제공
- 쿼리 최적화 기능 추가

## ■ Spark V3

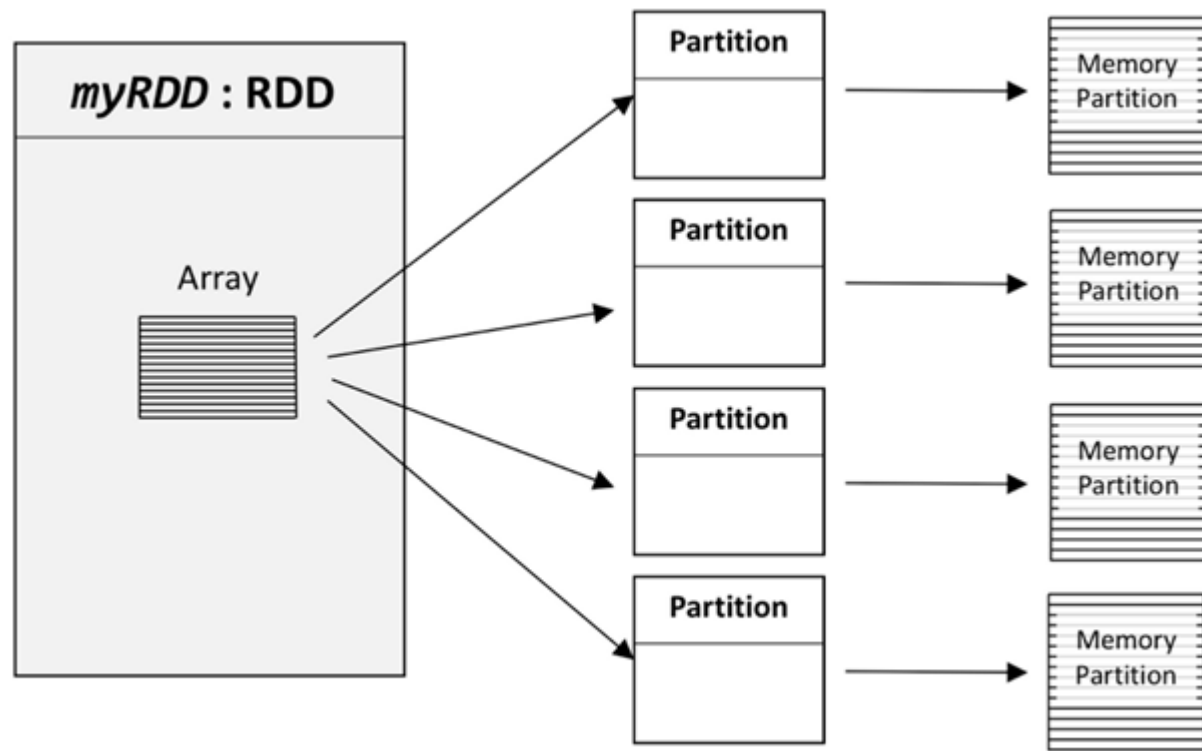
- 파이썬3, 스칼라 2.12, JDK 11 지원
- 딥러닝 지원 강화 : GPU 지원 추가
- 바이너리 파일 지원
- 쿠버네티스 지원 강화
- 다이나믹 파티션 프루닝(DPP) 지원하여 SQL 지원 강화

# RDD (Resilient Distributed Dataset)

- Spark에서 사용되는 가장 기본적인 데이터 객체입니다.
- Spark에서 데이터는 클러스터 메모리에 분산되어 Partition 단위로 분산 저장됩니다.
- Lineage(RDD를 만드는 일련의 단계)를 기록하여 노드의 장애/실패 발생 시 데이터를 재구성할 수 있습니다.
- RDD는 외부 데이터를 읽어서 처리하거나, 자체적으로 컬렉션 데이터를 생성하여 처리할 수 있습니다.

## ■ RDD 주요 특성(feature)

- Distributed Collection of Data
- Fault-tolerant
- Parallel operation – partitioned
- Ability to use many data sources



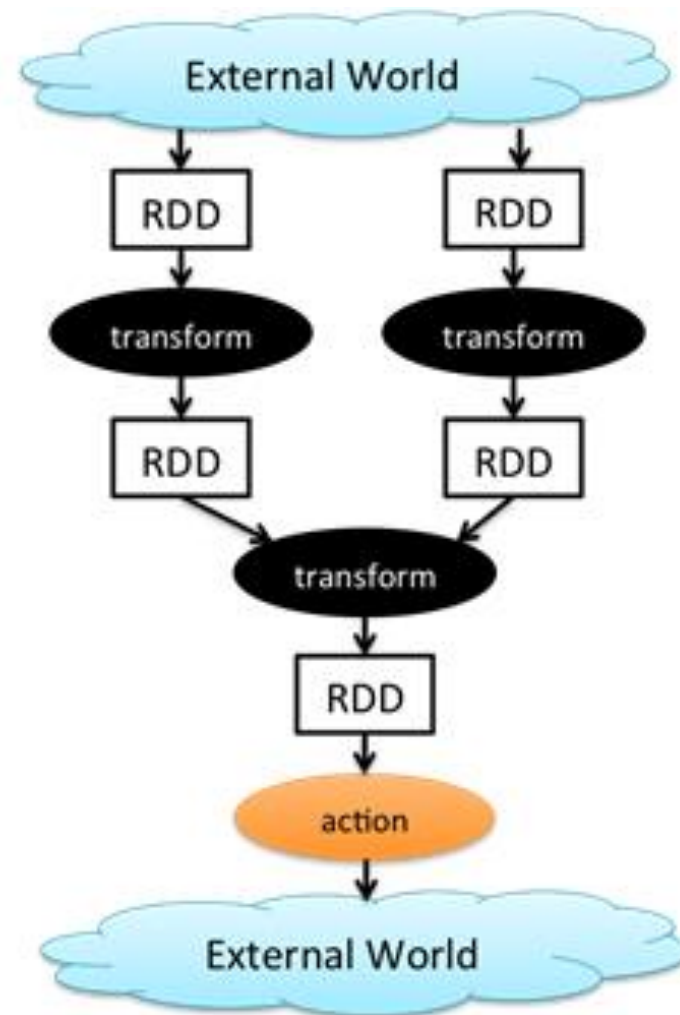
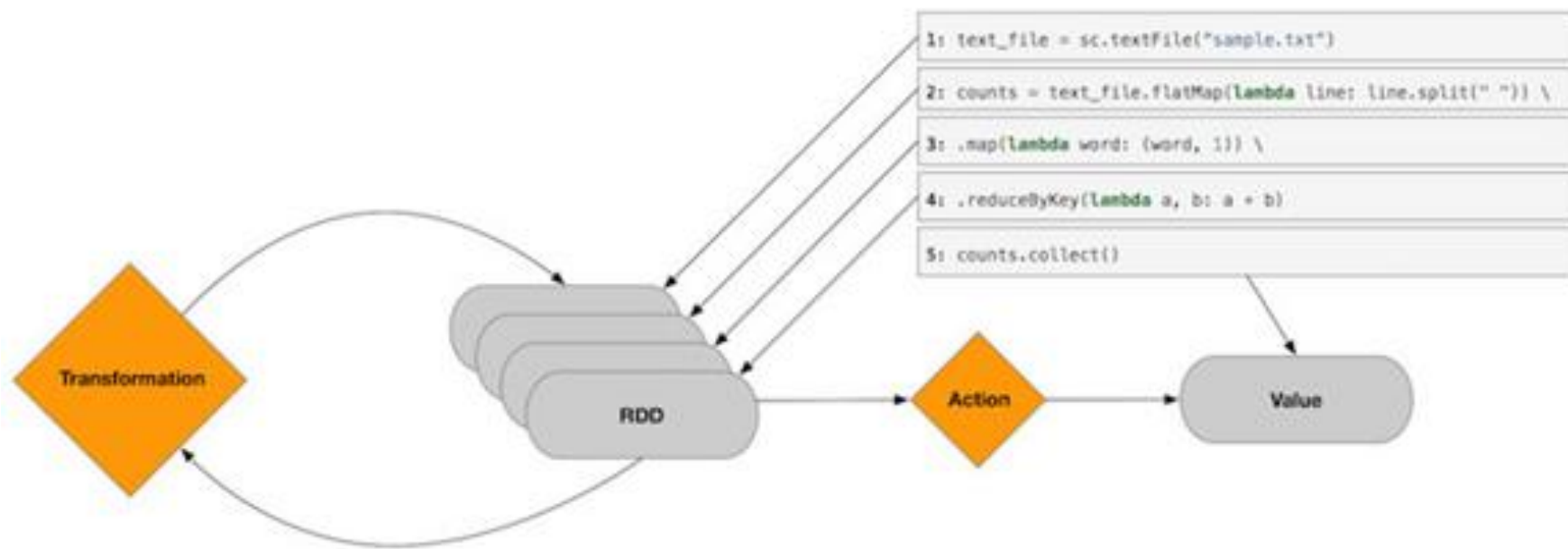
# RDD Operation

## Transformation

- Spark의 동작 중에서 데이터를 처리하는 명령
- map, filter, flatMap, join 등

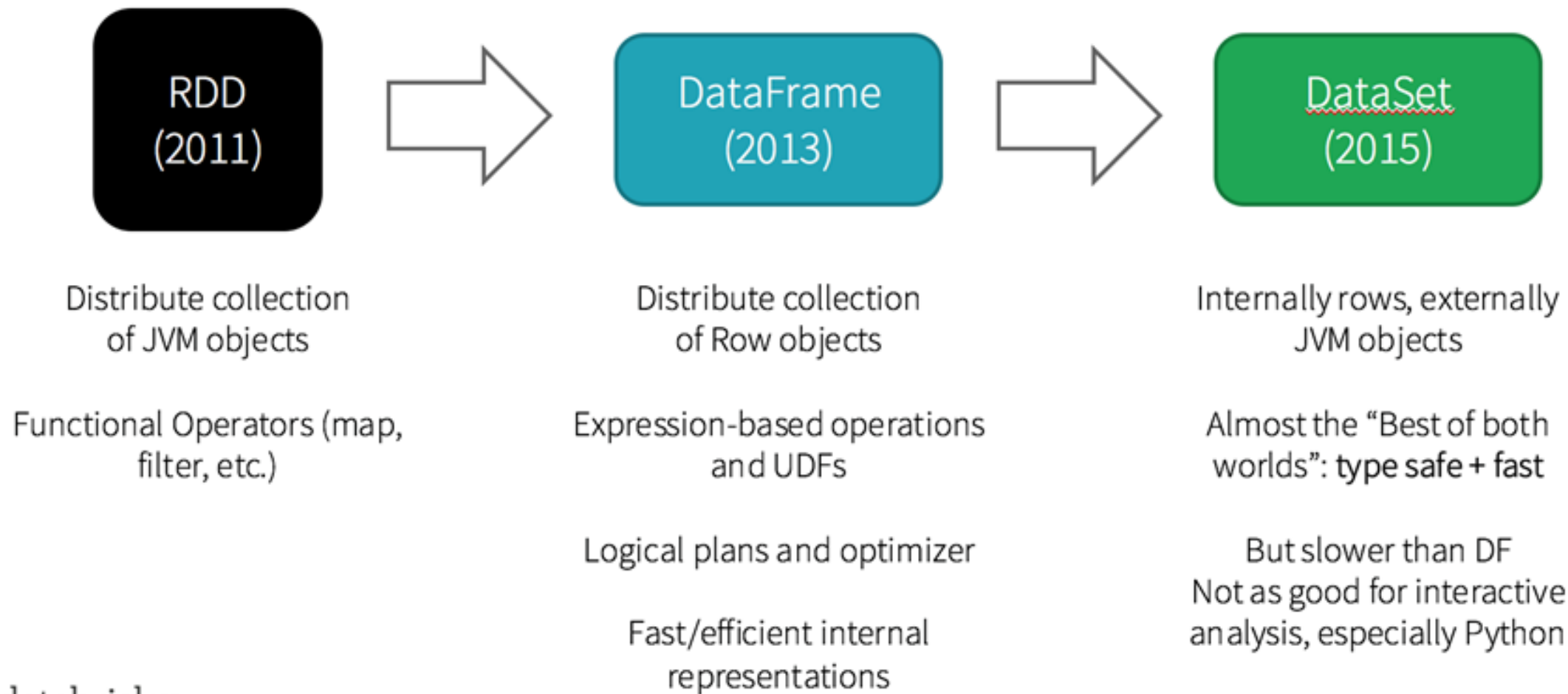
## Action

- Transformation의 결과를 저장하는 명령
- count, collect, reduce, save 등



# RDD

- Spark 1에서는 RDD를 주로 사용하였지만, Spark 2에서는 DataFrame(구조적 데이터 형태를) 사용





# Crerating RDD

- `nums = parallelize([1, 2, 3, 4])`
- `sc.textFile("file:///opt/spark/README.md")` or `s3n://` . `hdfs://`
- `hiveCtx = HiveContext(sc)` `rows = hiveCtx.sql("SELECT name, age FROM users")`
- Can also create from:
  - JDBC
  - Cassandra
  - Hbase
  - Elasticsearch
  - JSON, CSV, sequence files, object files, various compressed formats

# Transforming RDD

- map
- flatmap
- filter
- distinct
- sample
- union, intersection, subtract, cartesian

## ■ map example

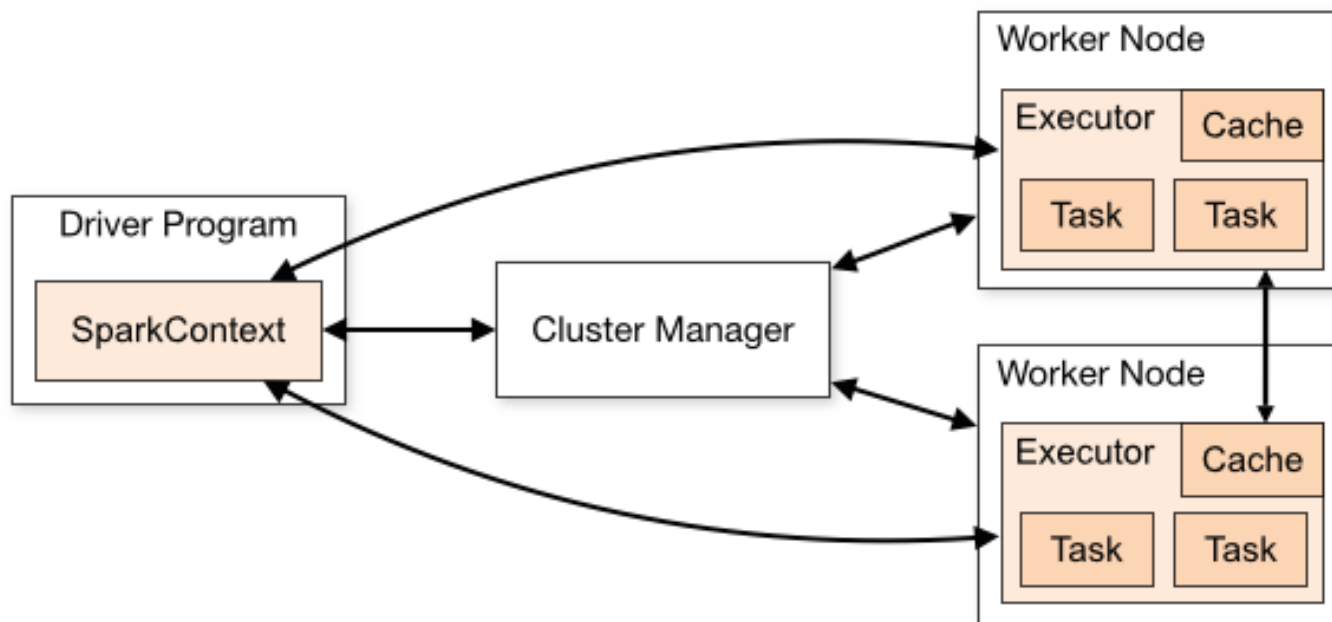
```
rdd = sc.parallelize([1, 2, 3, 4])  
quaredRDD = rdd.map(Lambda x: x*3)  
  
This yeilds 1, 4, 9, 16
```

# RDD actions

- collect
- count
- countByValue
- take
- top
- reduce
- ... and more ...

# Spark 애플리케이션 구조

- 스파크는 크게 두가지로 구성됩니다. 작업을 관리하는 드라이버 프로그램과 작업이 실행되는 노드를 관리하는 클러스터 매니저입니다. 스파크 애플리케이션은 마스터-슬레이브 구조로 실행됩니다.
- 스파크 애플리케이션은 작업을 관장하는 드라이버와 실제 작업이 동작하는 익스큐터로 구성됩니다.
- 드라이버는 스파크 컨텍스트 객체를 생성하여 클러스터 매니저와 통신하면서 클러스터의 자원 관리를 지원하고, 애플리케이션의 라이프 사이클을 관리합니다.
- Executor은 사용자가 만든 SparkContext(일종의 앱)를 위해 데이터를 저장하거나 연산을 실행하는 프로세스입니다.
- Spark는 Python, R, Java와 같은 프로그래밍 언어로 코드를 작성한 파일을 각 클러스터 내부에 있는 익스큐터들에게 전달합니다.



# Spark 애플리케이션

드라이버와 익스큐터 프로세스로 실행되는 프로그램으로  
클러스터 매니저가 스파크 애플리케이션의 리소스를 효율적으로 배분하게 됩니다.

## ■ 드라이버(Driver)

- 스파크 드라이버는 스파크 애플리케이션을 실행하는 프로세스입니다.
- main 함수를 실행하고, 스파크 컨텍스트(SparkContext) 객체를 생성합니다.
- 스파크 애플리케이션의 라이프 사이클을 관리하고, 사용자로 부터 입력을 받아서 애플리케이션에 전달합니다. 작업 처리 결과를 사용자에게 알려줍니다.
- 드라이버는 실행 시점에 디플로이 모드를 클라이언트 모드와 클러스터 모드로 설정할 수 있습니다.

## ■ 익스큐터(Executor)

- 태스크 실행을 담당하는 에이전트로 실제 작업을 진행하는 프로세스입니다.
- YARN의 컨테이너 라고 볼 수 있습니다.
- 익스큐터는 태스크 단위로 작업을 실행하고 결과를 드라이버에 알려줍니다.
- 익스큐터가 동작 중 오류가 발생하면 다시 재작업을 진행합니다.

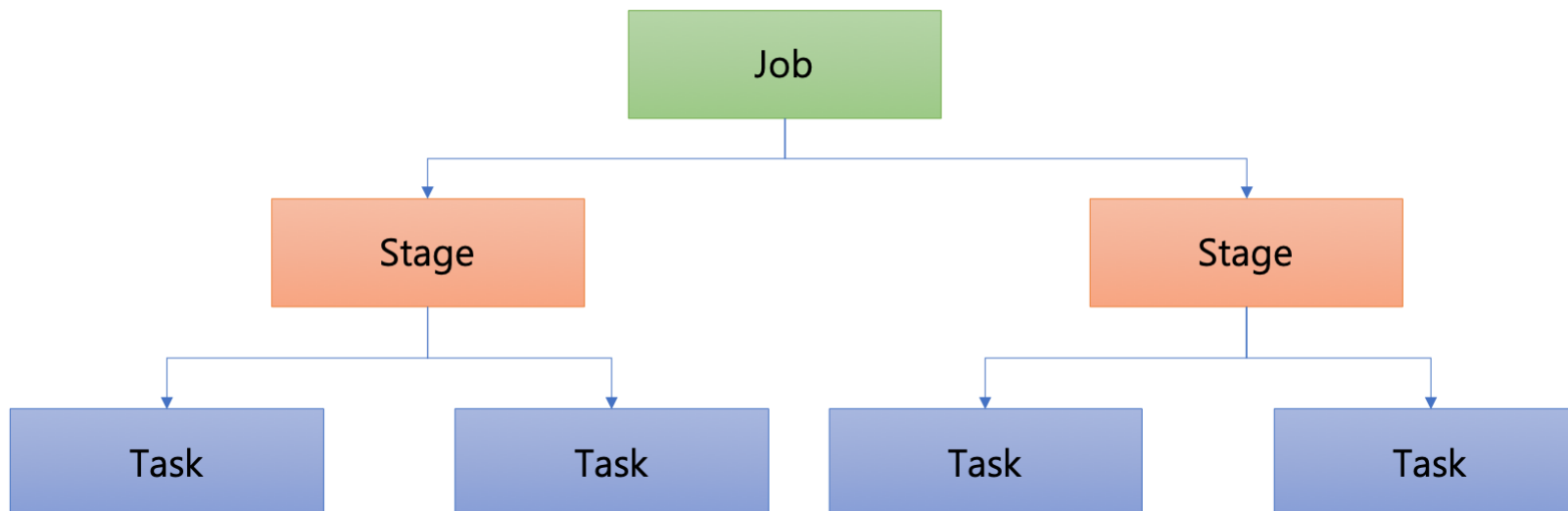
## ■ 태스크(Task)

- 익스큐터에서 실행되는 실제 작업입니다.
- 익스큐터의 캐쉬를 공유하여 작업의 속도를 높일 수 있습니다.

# Spark Job 구성

스파크 애플리케이션의 작업은 잡(Job), 스테이지(Stage), 태스크(Task)로 구성됩니다.

- 잡(Job) : 스파크 애플리케이션으로 제출된 작업입니다.
- 스테이지(Stage) : 잡을 작업의 단위에 따라 구분한 것이 스테이지입니다.
- 태스크(Task) : 익스큐터에서 실행되는 실제 작업입니다.  
데이터를 읽거나, 필터링 하는 실제 작업을 처리합니다.



# 클러스터 매니저

## ■ StandAlone

- 간단한 클러스터 매니저가 Spark에 포함되어있어 관리자는 클러스터를 쉽게 구성할 수 있습니다.
- 단일 머신에서 테스트를 위해 사용할 수 있습니다

## ■ Apache Mesos

- Hadoop의 MapReduce와 서비스 응용프로그램을 실행할 수 있는 일반적인 클러스터 관리자입니다.
- Spark와 다른 프레임워크 사이의 동적 구분과 Spark의 여러 인스턴스 사이의 확장 파티션 장점이 있습니다.

## ■ YARN

- YARN은 기존 MapReduce 중에서 클러스터의 리소스를 관리하는 부분만 가져와서 다른 서비스에서도 사용 가능 하도록 구성한 시스템입니다.
- YARN 시스템은 기존 MapReduce 보다는 기능적으로는 간단하지만, 범용적인 분산 리소스 관리 시스템이 되어야 했기 때문에 MapReduce의 클러스터 관리체계 보다 더 복잡하고 다양한 기능을 제공하고 있습니다.
- YARN은 핵심 구성 요소는 Resource Manager와 Node Manager입니다.
- 참조 : <https://www.popit.kr/what-is-hadoop-yarn/>

# Spark DataFrame

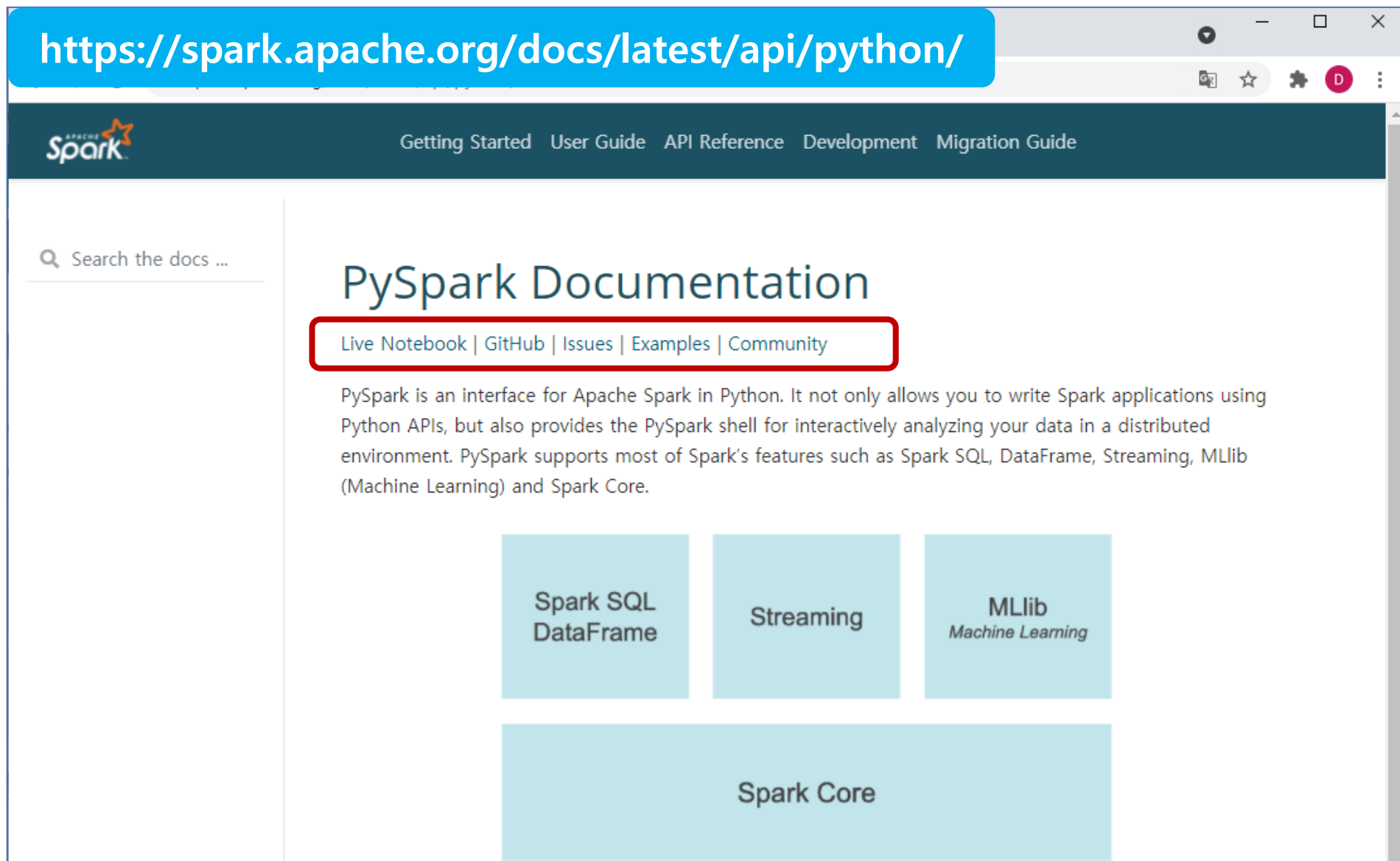
**Spark DataFrame은 Spark의 Machine Learning을 사용하는 표준 방법입니다,**

- Spark DataFrame은 column과 row 형식으로 데이터를 보유합니다.
- 각 column은 속성(feature)을 나타내며, 각 행은 개별 데이터 포인터입니다.
- RDD syntax는 Spark 2에서 훨씬 더 깔끔하고 작업하기 쉬운 DataFrame 구문으로 전환 되었습니다.
- Spark DataFrame은 다양한 소스의 데이터를 입력 및 출력 할 수 있습니다.
- DataFrame을 사용하여 데이터에 다양한 변환(transformation)을 적용 할 수 있습니다.



# PySpark

<https://spark.apache.org/docs/latest/api/python/>



The screenshot shows the Apache Spark documentation website for PySpark. The browser's address bar displays the URL <https://spark.apache.org/docs/latest/api/python/>. The page features a dark blue header with the Apache Spark logo and navigation links: Getting Started, User Guide, API Reference, Development, and Migration Guide. A search bar on the left is labeled "Search the docs ...". The main content area is titled "PySpark Documentation" and includes a red-bordered box with links: [Live Notebook](#) | [GitHub](#) | [Issues](#) | [Examples](#) | [Community](#). Below this, a paragraph describes PySpark as an interface for Apache Spark in Python, supporting features like Spark SQL, DataFrame, Streaming, MLlib (Machine Learning), and Spark Core. At the bottom, a diagram shows three light blue boxes labeled "Spark SQL DataFrame", "Streaming", and "MLlib Machine Learning" arranged horizontally above a larger light blue box labeled "Spark Core".

PySpark Documentation

[Live Notebook](#) | [GitHub](#) | [Issues](#) | [Examples](#) | [Community](#)

PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark's features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core.

Spark SQL  
DataFrame

Streaming

MLlib  
*Machine Learning*

Spark Core

# PySpark 실습(터미널 환경)

LowestRatedMovieSpark.py

Find the movie with the lowest average rating

```
1  from pyspark import SparkConf, SparkContext
2
3  # This function just creates a Python "dictionary" we can later
4  # use to convert movie ID's to movie names while printing out
5  # the final results.
6  def loadMovieNames():
7      movieNames = {}
8      with open("ml-100k/u.item") as f:
9          for line in f:
10             fields = line.split('|')
11             movieNames[int(fields[0])] = fields[1]
12     return movieNames
13
14 # Take each line of u.data and convert it to (movieID, (rating, 1.0))
15 # This way we can then add up all the ratings for each movie, and
16 # the total number of ratings for each movie (which lets us compute the average)
17 def parseInput(line):
18     fields = line.split()
19     return (int(fields[1]), (float(fields[2]), 1.0))
20
21 if __name__ == "__main__":
22     # The main script - create our SparkContext
23     conf = SparkConf().setAppName("WorstMovies")
24     sc = SparkContext(conf = conf)
25
```

LowestRatedMovieSpark.py

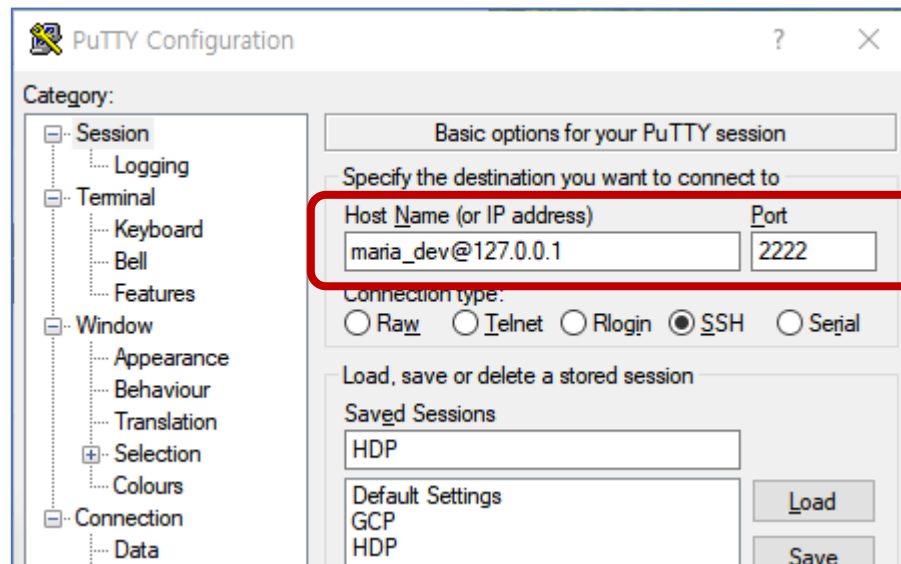
# PySpark 실습(터미널 환경)

```
26 # Load up our movie ID -> movie name lookup table
27 movieNames = loadMovieNames()
28
29 # Load up the raw u.data file
30 lines = sc.textFile("hdfs:///user/maria_dev/ml-100k/u.data")
31
32 # Convert to (movieID, (rating, 1.0))
33 movieRatings = lines.map(parseInput)
34
35 # Reduce to (movieID, (sumOfRatings, totalRatings))
36 ratingTotalsAndCount = movieRatings.reduceByKey(lambda movie1, movie2: ( movie1[0] + movie2[0], movie1[1] + movie2[1] ) )
37
38 # Map to (movieID, averageRating)
39 averageRatings = ratingTotalsAndCount.mapValues(lambda totalAndCount : totalAndCount[0] / totalAndCount[1])
40
41 # Sort by average rating
42 sortedMovies = averageRatings.sortBy(lambda x: x[1])
43
44 # Take the top 10 results
45 results = sortedMovies.take(10)
46
47 # Print them out:
48 for result in results:
49     print(movieNames[result[0]], result[1])
50
```

# PySpark 실습(터미널 환경)



putty 실행



## ■ HDP 터미널에서 명령어 실행

```
mkdir ml-100k
```

```
cd ml-100k
```

```
wget https://github.com/kgpark88/bigdata/raw/main/ml-100k/u.item
```

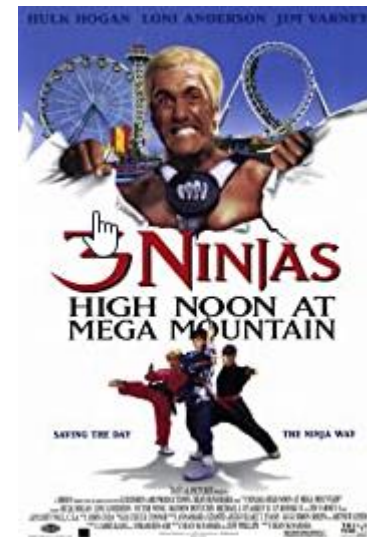
```
cd ..
```

```
wget https://github.com/kgpark88/bigdata/raw/main/LowestRatedMovieSpark.py
```

```
spark-submit LowestRatedMovieSpark.py
```

# PySpark 실습(터미널 환경)

```
maria_dev@sandbox-hdp:~  
21/06/06 05:35:15 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (es  
21/06/06 05:35:15 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on sandbox-hdp  
21/06/06 05:35:15 INFO SparkContext: Created broadcast 5 from broadcast at DAGScheduler.sc  
21/06/06 05:35:15 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 6 (Python  
21/06/06 05:35:15 INFO TaskSchedulerImpl: Adding task set 6.0 with 1 tasks  
21/06/06 05:35:15 INFO TaskSetManager: Starting task 0.0 in stage 6.0 (TID 8, localhost, e  
21/06/06 05:35:15 INFO Executor: Running task 0.0 in stage 6.0 (TID 8)  
21/06/06 05:35:15 INFO ShuffleBlockFetcherIterator: Getting 2 non-empty blocks out of 2 bl  
21/06/06 05:35:15 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms  
21/06/06 05:35:15 INFO PythonRunner: Times: total = 43, boot = -25, init = 67, finish = 1  
21/06/06 05:35:15 INFO Executor: Finished task 0.0 in stage 6.0 (TID 8). 1773 bytes result  
21/06/06 05:35:15 INFO TaskSetManager: Finished task 0.0 in stage 6.0 (TID 8) in 54 ms on  
21/06/06 05:35:15 INFO TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have all comple  
21/06/06 05:35:15 INFO DAGScheduler: ResultStage 6 (runJob at PythonRDD.scala:141) finishe  
21/06/06 05:35:15 INFO DAGScheduler: Job 2 finished: runJob at PythonRDD.scala:141, took 0  
( '3 Ninjas: High Noon At Mega Mountain (1998)', 1.0)  
( 'Beyond Bedlam (1993)', 1.0)  
( 'Power 98 (1995)', 1.0)  
( 'Bloody Child, The (1996)', 1.0)  
( 'Amityville: Dollhouse (1996)', 1.0)  
( 'Babyfever (1994)', 1.0)  
( 'Homage (1995)', 1.0)  
( 'Somebody to Love (1994)', 1.0)  
( 'Crude Oasis, The (1995)', 1.0)  
( 'Every Other Weekend (1990)', 1.0)
```



# Colab에서 PySpark 사용하는 방법

spark\_in\_colab.ipynb

The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1mDiTzbBhcORCs7OcG7XVKZMIHxp2Y0EA#scrollTo=baECVey4xrv8`. The notebook title is `spark_in_colab.ipynb`. The interface includes a left sidebar with icons for file management, search, and code execution. The main area shows two sections: **Java Virtual Machine (JVM) 설치** and **Apache Spark 설치**. The first section contains a code cell with the command `!apt-get install openjdk-8-jdk-headless`. The second section contains three code cells with the following commands: `!wget -q https://downloads.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop2.7.tgz`, `!tar xf spark-3.1.2-bin-hadoop2.7.tgz`, and `!ls -lt`. The bottom right corner of the notebook window shows a close button.

spark\_in\_colab.ipynb - Colaboratory

colab.research.google.com/drive/1mDiTzbBhcORCs7OcG7XVKZMIHxp2Y0EA#scrollTo=baECVey4xrv8

spark\_in\_colab.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

연결 수정 가능

Java Virtual Machine (JVM) 설치

```
[ ] !apt-get install openjdk-8-jdk-headless
```

Apache Spark 설치

```
[ ] !wget -q https://downloads.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop2.7.tgz
```

```
[ ] !tar xf spark-3.1.2-bin-hadoop2.7.tgz
```

```
[ ] !ls -lt
```

# Colab에서 PySpark 사용하는 방법

spark\_in\_colab.ipynb

## findspark 라이브러리 설치

```
[ ] !pip install -q findspark
```

## 환경변수 설정

```
[ ] import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop2.7"
```

## 테스트

```
[ ] import findspark
    findspark.init()
```

```
[ ] findspark.find()
```

`'/content/spark-3.1.2-bin-hadoop2.7'`

# PySpark RDD 실습

spark\_rdd.ipynb

## ■ textFile() 메소드를 사용해 RDD 만들기

```
[7] # licenses RDD에 전체 디렉토리의 내용 로드  
    license_files = sc.textFile("file:///content/spark-3.1.2-bin-hadoop2.7/licenses/")
```

```
[8] # 생성된 객체 검사  
    license_files
```

```
file:///content/spark-3.1.2-bin-hadoop2.7/licenses/ MapPartitionsRDD[6] at textFile at NativeMethodAccessorImpl.java:0
```

```
[9] license_files.take(1)  
  
['JTransforms']
```

```
[10] license_files.getNumPartitions()
```

```
57
```

```
[11] # 모든 파일에서 총 라인 수  
    license_files.count()
```

```
2973
```



# PySpark RDD 실습

spark\_rdd.ipynb

## ■ 데이터 소스에서 RDD 생성

```
[13] people = spark.read.json("/content/spark-3.1.2-bin-hadoop2.7/examples/src/main/resources/people.json")
```

```
[14] people
```

```
DataFrame[age: bigint, name: string]
```

```
[15] people.dtypes
```

```
[('age', 'bigint'), ('name', 'string')]
```

```
[▶] people.show()
```

```
┌-----┬-----┐
│  age │  name │
├-----┴-----┤
│ null │ Michael │
│   30 │   Andy │
│   19 │  Justin │
└-----┴-----┘
```

```
[17] from pyspark.sql import SQLContext
```

```
# as with all DataFrames you can create use them to run SQL queries as follows
sqlContext = SQLContext(sc)
sqlContext.registerDataFrameAsTable(people, "people")
df2 = spark.sql("SELECT name, age FROM people WHERE age > 20")
df2.show()
```

```
┌-----┬-----┐
│ name │ age │
├-----┴-----┤
│ Andy │  30 │
└-----┴-----┘
```

# PySpark RDD 실습

spark\_rdd.ipynb

## ■ 프로그래밍 방식으로 RDD 생성

```
[18] parallelRdd = sc.parallelize([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
[19] parallelRdd
```

ParallelCollectionRDD[21] at readRDDFromFile at PythonRDD.scala:274

```
[20] parallelRdd.count()
```

9

```
[21] # 0에서 시작해서 1000개의 정수로, 2개의 파티션에서 1씩 증가하는 RDD 생성
range_rdd = sc.range(0, 1000, 1, 2)
range_rdd
```

PythonRDD[24] at RDD at PythonRDD.scala:53

```
[22] range_rdd.getNumPartitions()
```

2

# PySpark DataFrame 실습

spark\_dataframe.ipynb

## ■ 데이터 파일 : people.json

```
{"name":"Michael"}
```

```
{"name":"Andy", "age":30}
```

```
{"name":"Justin", "age":19}
```

## ■ Creating a DataFrame

```
[2] from pyspark.sql import SparkSession
```

```
[3] # May take a little while on a local computer  
     spark = SparkSession.builder.appName("Basics").getOrCreate()
```

```
[4] df = spark.read.json('people.json')
```

# PySpark DataFrame 실습

spark\_dataframe.ipynb

## ■ Showing the data

```
[5] # Note how data is missing!  
df.show()
```

```
+-----+-----+  
| age |   name |  
+-----+-----+  
| null | Michael |  
|   30 |   Andy |  
|   19 |  Justin |  
+-----+-----+
```

```
[6] df.printSchema()
```

```
root  
 |-- age: long (nullable = true)  
 |-- name: string (nullable = true)
```

```
[7] df.columns
```

```
['age', 'name']
```

```
[8] df.describe()
```

```
DataFrame[summary: string, age: string, name: string]
```

# PySpark DataFrame 실습

spark\_dataframe.ipynb

## ■ Infer schema

```
[9] from pyspark.sql.types import StructField,StringType,IntegerType,StructType
```

```
[10] data_schema = [StructField("age", IntegerType(), True),StructField("name", StringType(), True)]
```

```
[11] final_struct = StructType(fields=data_schema)
```

```
[12] df = spark.read.json('people.json', schema=final_struct)
```

```
[13] df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- name: string (nullable = true)
```

# PySpark DataFrame 실습

spark\_dataframe.ipynb

## Grabbing the data

```
[14] df['age']
```

```
Column<'age'>
```

```
[15] type(df['age'])
```

```
pyspark.sql.column.Column
```

```
[16] df.select('age')
```

```
DataFrame[age: int]
```

```
[17] type(df.select('age'))
```

```
pyspark.sql.dataframe.DataFrame
```

```
[18] df.select('age').show()
```

```
+-----+  
|  age  |  
+-----+  
| null  |  
|   30  |  
|   19  |  
+-----+
```

# PySpark DataFrame 실습

spark\_dataframe.ipynb

## ■ Multiple Columns

```
[20] df.select(['age', 'name'])
```

```
DataFrame[age: int, name: string]
```

```
[21] df.select(['age', 'name']).show()
```

```
+----+-----+
| age|   name|
+----+-----+
| null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-----+
```

## ■ Creating new columns

```
[22] # Adding a new column with a simple copy
df.withColumn('newage',df['age']).show()
```

```
+----+-----+-----+
| age|   name|newage|
+----+-----+-----+
| null|Michael|  null|
|  30|   Andy|   30|
|  19| Justin|   19|
+----+-----+-----+
```

# PySpark DataFrame 실습

spark\_dataframe.ipynb

## More complicated operations to create new columns

```
[25] df.withColumn('doubleage',df['age']*2).show()
```

age	name	doubleage
null	Michael	null
30	Andy	60
19	Justin	38

```
[26] df.withColumn('add_one_age',df['age']+1).show()
```

age	name	add_one_age
null	Michael	null
30	Andy	31
19	Justin	20

<https://sparkbyexamples.com/pyspark-tutorial/>



# PySpark ML 실습



`spark_linear_regression.ipynb`

`spark_logistic_regression.ipynb`

`spark_tree_model.ipynb`

# PySpark Tutorial

<https://sparkbyexamples.com/pyspark-tutorial/>

## What is PySpark

Introduction

Who uses PySpark

Features

Advantages

## PySpark Architecture

Cluster Manager Types

Modules and Packages

PySpark Installation on windows

Spyder IDE & Jupyter Notebook

## PySpark RDD

RDD creation

RDD operations

## PySpark DataFrame

Is PySpark faster than pandas?

DataFrame creation

DataFrame Operations

DataFrame external data sources

Supported file formats

## PySpark SQL

## PySpark Streaming

Streaming from TCP Socket

Streaming from Kafka

## PySpark GraphFrames

GraphX vs GraphFrames

# Thank you