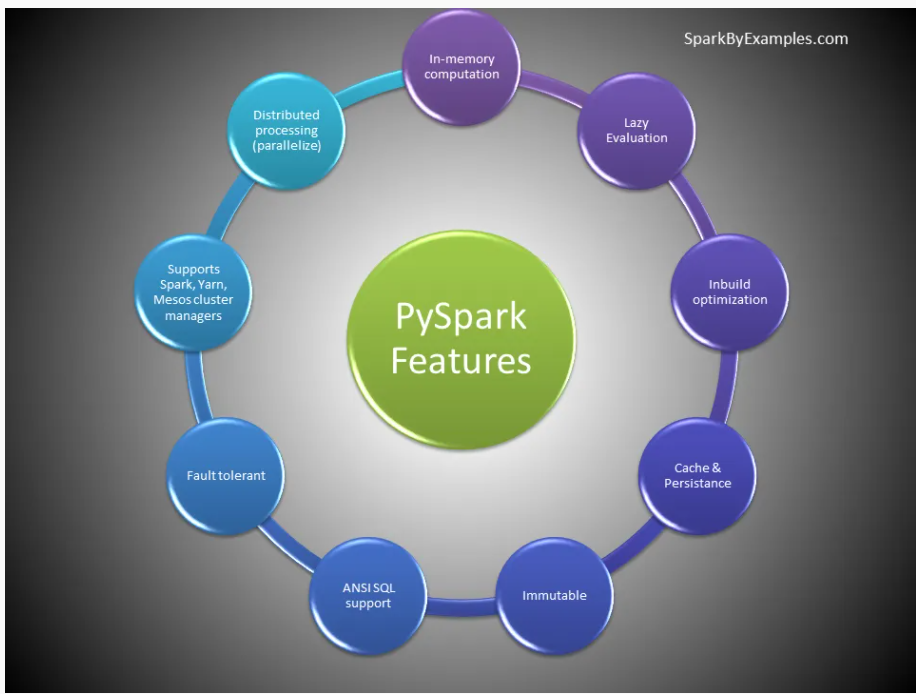
**BÀI TẬP 2: TÌM HIỂU SPARK**

1. **Spark properties**
   1. **Đặc trưng**

****

Tính toán trong bộ nhớ

Xử lý phân tán bằng sử dụng song song

Có thể được sử dụng với nhiều trình quản lý cụm(Spark, Yarn,…)

Chịu được lỗi

Bất biến

Bộ nhớ Cache và bền bỉ

Hỗ trợ ANSI SQL

Tối ưu hóa Inbuild khi sử dụng DataFrame

* 1. **PySpark Module và Packages**

PySpark RDD (pyspark.RDD)

PySpark DataFrame and SQL (pyspark.sql)

PySpark Streaming (pyspark.streaming)

PySpark MLib (pyspark.ml, pyspark.mllib)

PySpark GraphFrames (GraphFrames)

PySpark Resource (pyspark.resource) It’s new in PySpark 3.0

1. **Spark RDD**
   1. **Spark RDD là gì?**

RDD( Resilient Distributed Dataset) là một khối xây dựng cơ bản của PySpark, tập hợp các đối tượng phân tán không thay đổi. Không thể thay đổi RDD khi đã tạo ra nó. Mỗi bản ghi RDD được chia thành các phân vùng logic, có thể tính toán trên các nút khác nhau của cụm

Nói cách khác, RDD là một tập hợp các đối tượng tương tự như danh sách trong Python, với sự khác biệt là RDD được tính toán trên một số quy trình nằm rải rác trên nhiều máy chủ vật lý còn được gọi là các nút trong một cụm trong khi tập hợp Python tồn tại và xử lý chỉ trong một quy trình

Ngoài ra, RDD cung cấp sự trừu tượng hóa dữ liệu về phân vùng và phối phối dữ liệu được thiết kế để chạy tính toán song song trên một số nút

* 1. **Lợi ích của PySpark RDD**

Xử lí trong bộ nhớ:

* PySpark tải dữ liệu từ đĩa và xử lý trong bộ nhớ và giữ dữ liệu trong bộ nhớ, đây là điểm khác biệt chính giữa PySpark và Mapreduce (I / O chuyên sâu). Giữa các lần biến đổi, chúng ta cũng có thể lưu cache / duy trì RDD trong bộ nhớ để sử dụng lại các tính toán trước đó.

Tính bất biến:

* Một khi đã tạo ra RDD thì không thể thay đổi được nó

Chịu lỗi:

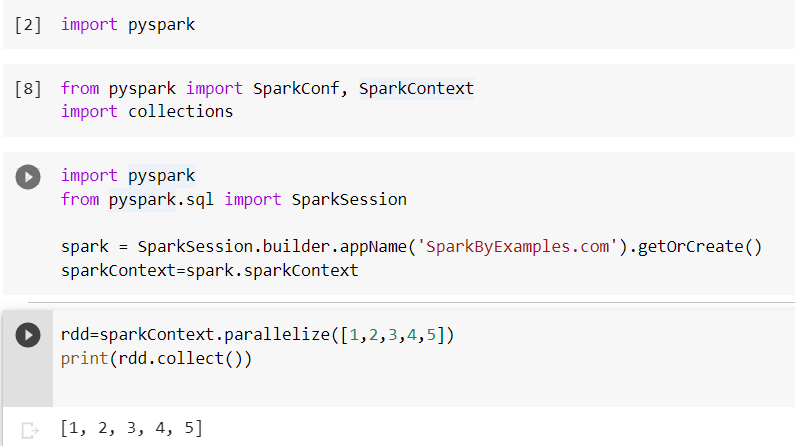
* PySpark hoạt động trên các kho dữ liệu có khả năng chịu lỗi trên HDFS, S3 e.t.c, do đó bất kỳ hoạt động RDD nào không thành công, nó sẽ tự động tải lại dữ liệu từ các phân vùng khác. Ngoài ra, Khi các ứng dụng PySpark chạy trên một cụm, các lỗi tác vụ PySpark sẽ tự động được khôi phục trong một số lần nhất định (theo cấu hình) và kết thúc ứng dụng một cách liền mạch.

Lazy Evolution:

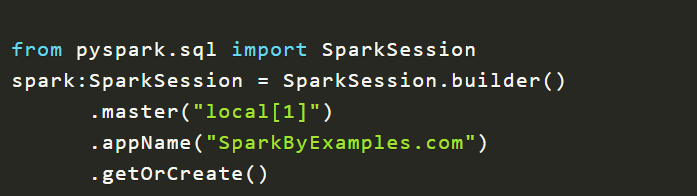
* PySpark không đánh giá các phép biến đổi RDD khi chúng xuất hiện / gặp phải bởi Driver thay vào đó nó giữ tất cả các phép biến đổi khi nó gặp (DAG) và đánh giá tất cả các phép biến đổi khi nó thấy hành động RDD đầu tiên

Phân vùng:

* Khi tạo RDD từ một dữ liệu, nó theo mặc định phân vùng các phần tử trong RDD. Theo mặc định, nó phân vùng theo số lượng lõi có sẵn.
  1. **Tạo RDD**

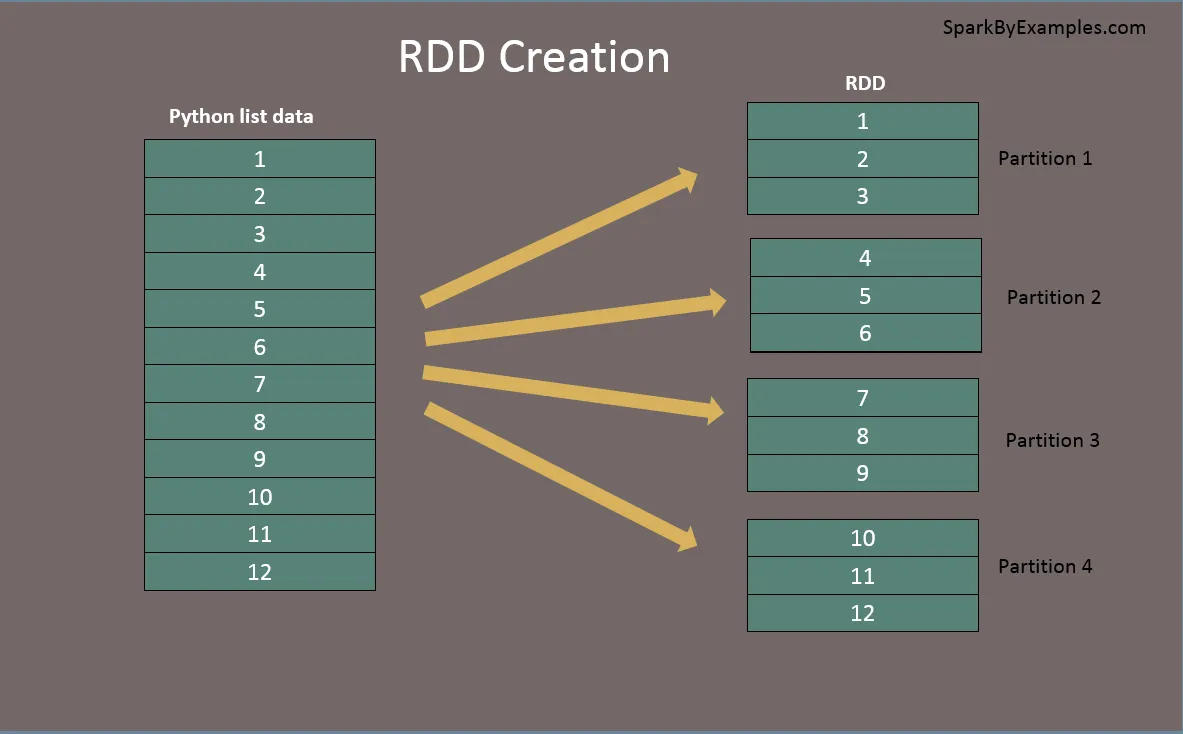


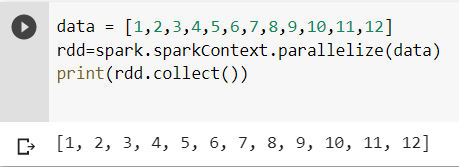
Bước đầu cần khởi tạo SparkSession bằng cách sử dụng phương thức mẫu, cần cung cấp tên master và app như hình, trong đó:



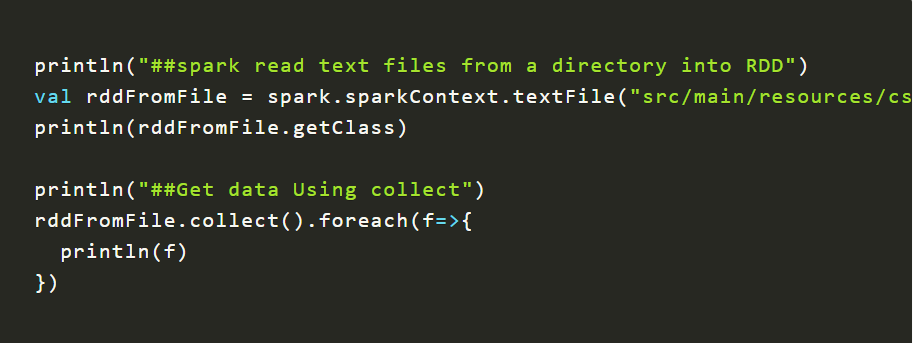
* Master() – Nếu chạy nó trên cụm, cần sử dụng tên chính của mình làm đối số cho master(), thông thường sẽ là yarn hoặc mesos tùy vào thiết lập
  + Sử dụng local[x] khi chạy ở chế độ độc lập. x là một giá trị nguyên dương, thể hiện số lượng phân vùng nó sẽ tạo khi sử dụng RDD, Dataframe và Dataset. Tốt nhất thì x nên là số lõi CPU
* appName() – Tên ứng dụng
* getOrCreate() – Trả về một đối tượng SparkSession nếu đã tồn tại hoặc tạo một đối tượng mới nếu chưa tồn tại

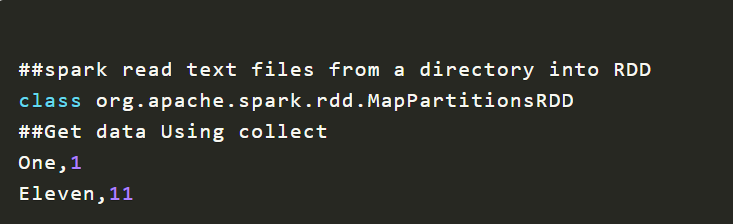
**Tạo RDD bằng sparkContext.parallelize()**



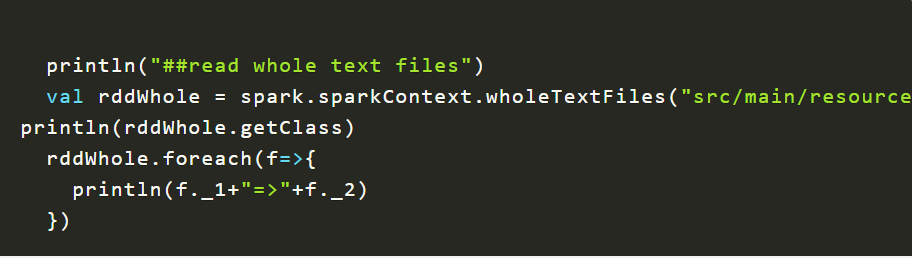
****

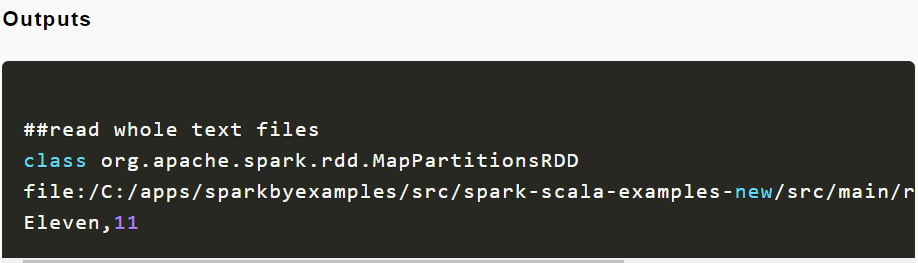
**Tạo RDD bằng sparkContext.textFile()**

****

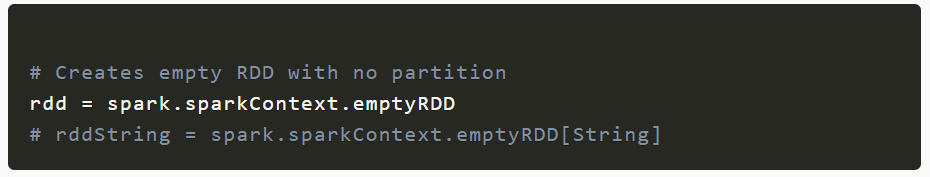


**Tạo RDD bằng wholeTextFiles**

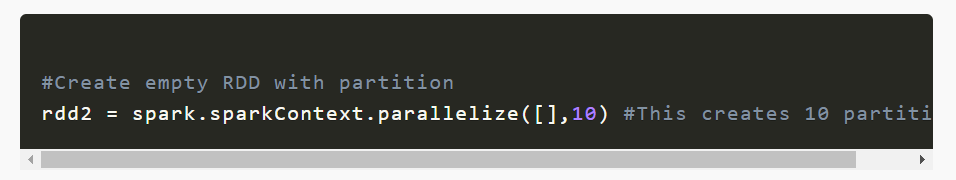
****

****

**Tạo RDD rỗng bằng sparkContext.emptyRDD**

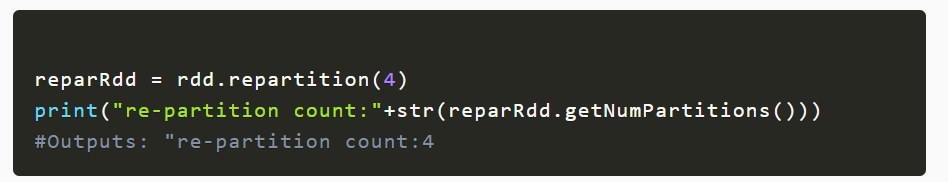
****

**Tạo RDD rỗng bằng vách ngăn**

****

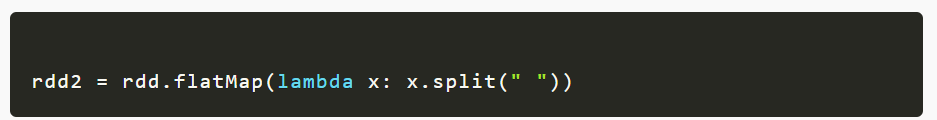
* 1. **Phân vùng và kết hợp**

Pyspark cung cấp hai cách để phân vùng, cách thứ nhất sử dụng repartition(), cách còn lại là Coalesce

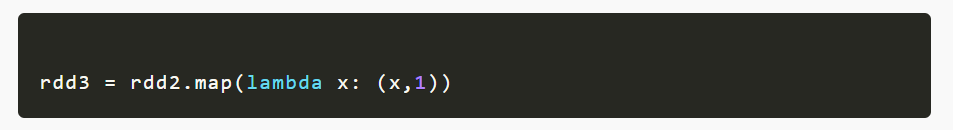


* 1. **Các phép biến đổi của RDD**

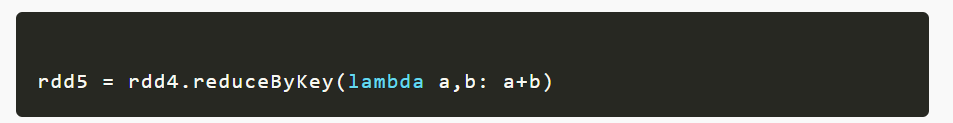
**Flatmap():**



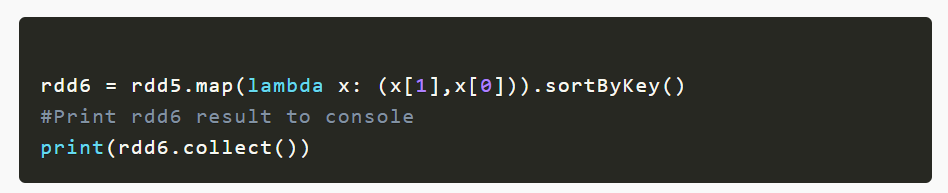
**Map():**



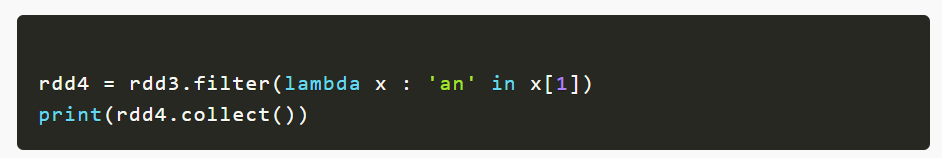
**ReduceKey():**

****

**SortByKey():**

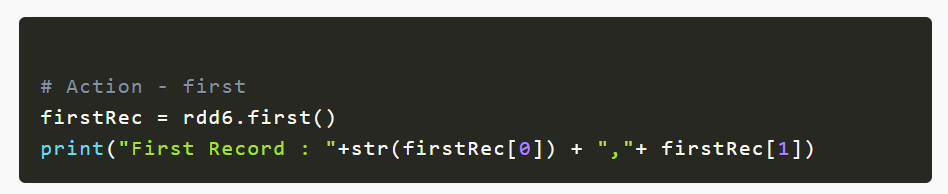
****

**Filter():**

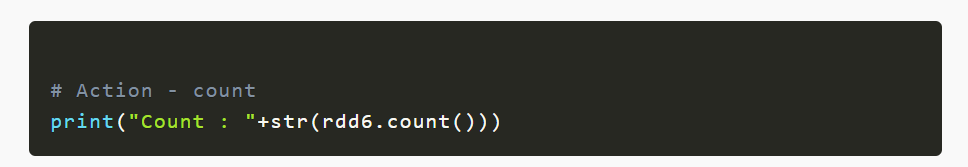
****

* 1. **Các action của RDD**

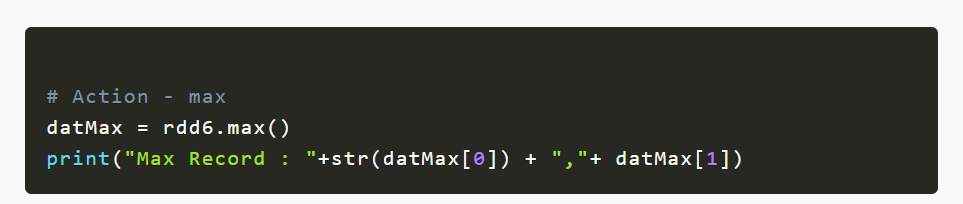
**First()**

****

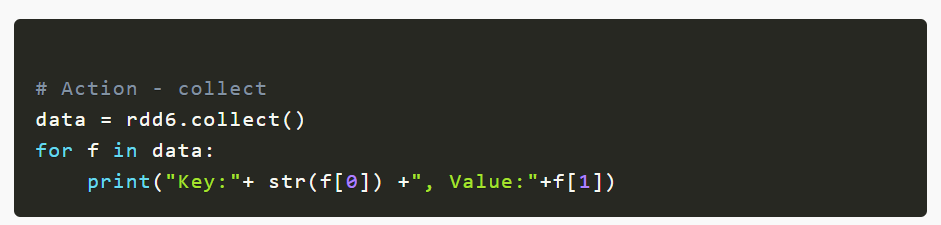
**Count()**

****

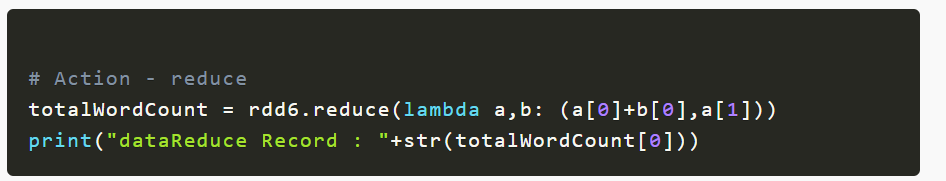
**Max()**

****

**Collect()**

****

**Reduce()**

****

* 1. **Các loại RDD**

PairRDDFunctions or PairRDD

ShuffledRDD

DoubleRDD

SequenceFileRDD

HadoopRDD

ParallelCollectionRDD

* 1. **RDD Cache**

Lưu tính toán RDD ở mức lưu trữ ‘MEMORY\_ONLY’ nghĩa là nó sẽ lưu trữ trong heap JVM dưới dạng các đối tượng chưa được quản lý



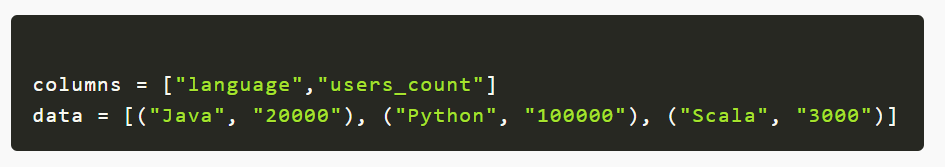
1. **Spark DataFrame**
   1. **Tạo DataFrame**

Có thể tạo DataFrame bằng toDF() and create DataFrame()

Có thể tạo DataFrame từ nhiều nguồn dữ liệu như txt, csv, json, … định dạng XML bằng cách đọc từ hệ thống tệp HDFS, S3, DBFS, …

DataFrame cũng có thể được tạo bằng cách đọc dữ liệu từ Cơ sở dữ liệu RDBMS và Cơ sở dữ liệu NoSQL.

Để tạo Dataframe từ danh sách, cần tạo dữ liệu và các cột cần thiết



* + 1. Tạo Dataframe từ RDD

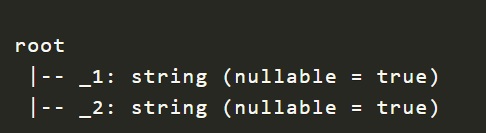
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

rdd = spark.sparkContext.parallelize(data)

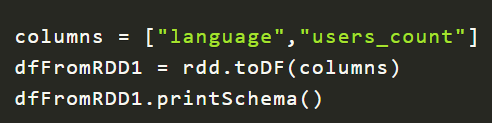
**Sử dụng hàm toDF():**



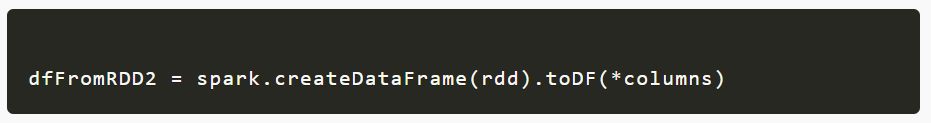
PrintSchema() sẽ được:



Tạo tên cột cho Dataframe:



**Sử dụng createDataFrame() từ SparkSession:**

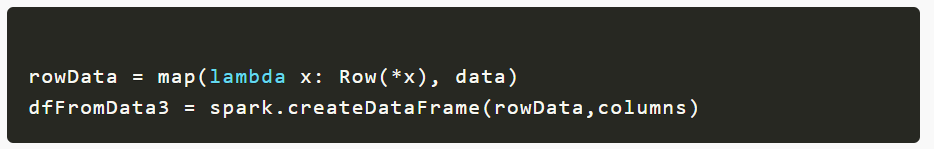
****

* + 1. Tạo Dataframe từ List Collection

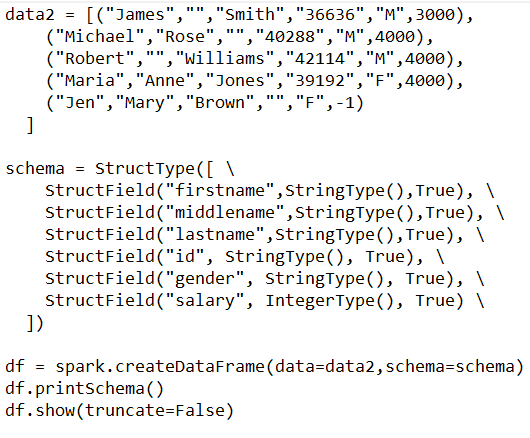
**Sử dụng createDataFrame() từ SparkSession:**

****

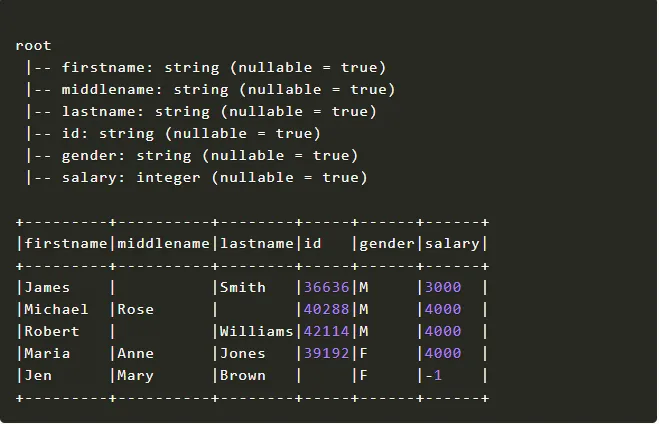
**Sử dụng createDataFrame() với Row**

****

**Tạo DataFrame với schema**

****

Output:



* + 1. Tạo Dataframe từ nguồn dữ liệu

**Tạo DataFrame từ CSV:**

Sử dụng phương thức csv() để đối tượng DataFrameReader để tạo DataFrame từ file CSV



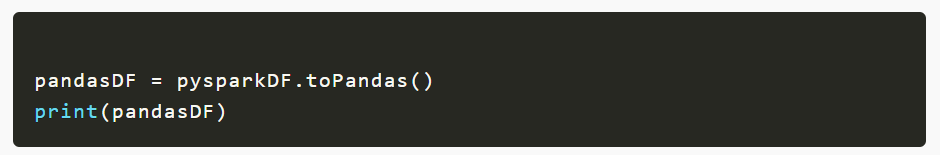
**Tạo DataFrame từ file text:**

****

**Tạo DataFrame từ JSON:**

****

* 1. **Chuyển PySpark DataFrame sang Pandas DataFrame**

****