Data Engineering tool : Databrick & Pyspark

Course Outline

1. What is Databricks
2. Databricks Components
3. What is Pyspark
4. Install Pyspark using Python for Jupyter Notebook
5. Register Databricks Community Edition
6. Basic Python
7. Databricks DBUTILS Command
8. Pyspark Basic Command

1. WHAT IS DATABRICKS

Databricks is a unified set of tools for building, deploying, sharing, and maintaining enterprise-grade data solutions.

Databricks has been developed to cover the most applications by divided into 3 environments :

- Databricks Data Science & Engineering

- Databricks Machine Learning

  Databricks Machine Learning empowers ML teams to prepare and process data, streamlines cross-team collaboration and standardizes the full ML lifecycle from experimentation to production.

- Databricks SQL

  serverless data warehouse on the Databricks Lakehouse Platform that lets you run all your SQL and BI applications at scale

databricks

1.1 Databricks Data Science & Engineering

Databricks Data Science & Engineering is the classic **Databricks environment for collaboration among data scientists, data engineers, and data analysts**.

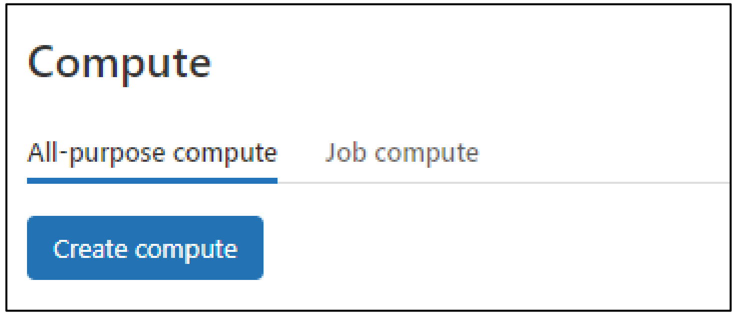
Data Science & Engineering Learning Guide :

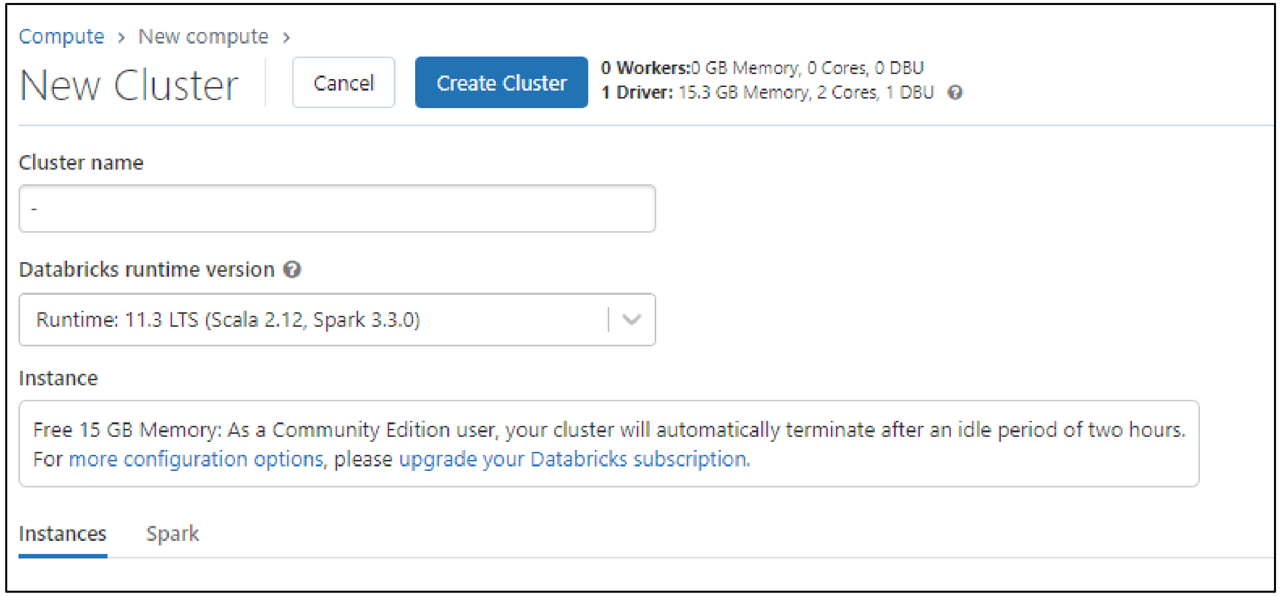
- Delta Live Tables
- Apache Spark
- Cluster
- Workflows
- Libraries
- DBFS
- Migration
- Structured Streaming
- Runtimes
- Notebooks
- Storage
- Repos
- Files
- Optimization & performance

2. (1) DATABRICKS COMPONENTS : CLUSTER

A Databricks cluster is a set of **computation resources and configurations on which you run data engineering, data science, and data analytics workloads**, such as production ETL pipelines, streaming analytics, ad-hoc analytics, and machine learning.

You run these workloads as a set of commands in a notebook or as an automated job. Databricks makes a distinction between all-purpose clusters and job clusters. You use all-purpose clusters to analyze data collaboratively using interactive notebooks.





0 Workers:0 GB Memory, 0 Cores, 0 DBU
1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU

Cluster name

-

Databricks runtime version

Runtime: 11.3 LTS (Scala 2.12, Spark 3.3.0)

Instance

Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For more configuration options, please upgrade your Databricks subscription.

2. (2) DATABRICKS COMPONENTS : NOTEBOOKS

Notebooks are a common tool in data science and machine learning for developing code and presenting results. In Databricks, notebooks are the primary tool for creating data science and machine learning workflows and collaborating with colleagues. Databricks notebooks provide real-time coauthoring in multiple languages, automatic versioning, and built-in data visualizations.

With Databricks notebooks :

- Develop code using **Python**, **SQL**, **Scala**, and **R**.

- **Customize your environment with the libraries** of your choice.

- **Create regularly scheduled jobs to automatically run tasks**, including multi-notebook workflows.

- **Export results and notebooks in .html or .ipynb format**.

- Use a **Git-based repository to store your notebooks** with associated files and dependencies.

- **Build and share dashboards**.

- **Open or run a Delta Live Tables pipeline**.

2. (3) DATABRICKS COMPONENTS : STORAGE

Databricks uses a shared responsibility model to create, configure, and access **block storage** volumes and **object storage** locations in your cloud account. Loading data to or saving data with Databricks results in files stored in either block storage or object storage.

Operation	**Location**
UI data upload	Object storage
DBFS file upload	Object storage
Upload data with Auto Loader	Object storage
Upload data with COPY INTO	Object storage
Create table	Object storage
Save data with Apache Spark	Object storage
Save data with pandas	Block storage
Download data from web in a notebook	Block storage

2.3.1 Object Storage

Object storage or blob storage refers to **storage containers that maintain data as objects, with each object consisting of data, metadata, and a globally unique resource identifier** (URI). Data manipulation operations in object storage are often limited to create, read, update, and delete (CRUD) through a REST API interface. Some object storage offerings include features like versioning and lifecycle management.

Object storage has the following benefits:

- High availability, durability, and reliability.

- Lower cost for storage compared to most other storage options.

- Infinitely scalable (limited by the total amount of storage available in a given region of the cloud).

In Databricks , Object storage is the main form of storage used by Databricks for most operations. The Databricks Filesystem (DBFS) allows Databricks users to interact with files in object storage similarly to how they would in any other file system. Unless you specifically configure a table against an external data system, all tables created in Databricks store data in cloud object storage.

2.3.2 Block Storage

Block storage or disk storage refer to storage volumes that correspond to traditional hard disk drives (HDDs) or solid state drives (SSDs), also known simply as “hard drives”. When deploying block storage in a cloud computing environment, typically a logical partition of one or more physical drives are deployed. Implementations vary slightly between product offerings and cloud vendors, but the following characteristics are typically found across implementations:

All virtual machines (VMs) require an attached block storage volume.

- Files and programs installed to a block storage volume persist as long as the block storage volume persists.

- **Block storage volumes are often used for temporary data storage**.

- Block storage volumes attached to VMs are usually deleted alongside VMs.

Databricks configures and deploys VMs and attaches block storage volumes**. This block storage is used for storing ephemeral data files for the lifetime of the compute.** These files include the operating system and installed libraries, in addition to data used by the disk cache. While **Apache Spark uses block storage in the background for efficient parallelization and data loading**, most code run on Databricks does not directly save or load data to block storage.

3. Pyspark

Apache Spark is written in Scala programming language. **PySpark has been released in order to support the collaboration of Apache Spark and Python**.With PySpark, you can write Python and SQL-like commands to manipulate and analyze data in a distributed processing environment.

Also data that store in Pyspark variable store as **Resilient Distributed Datasets (RDD)**

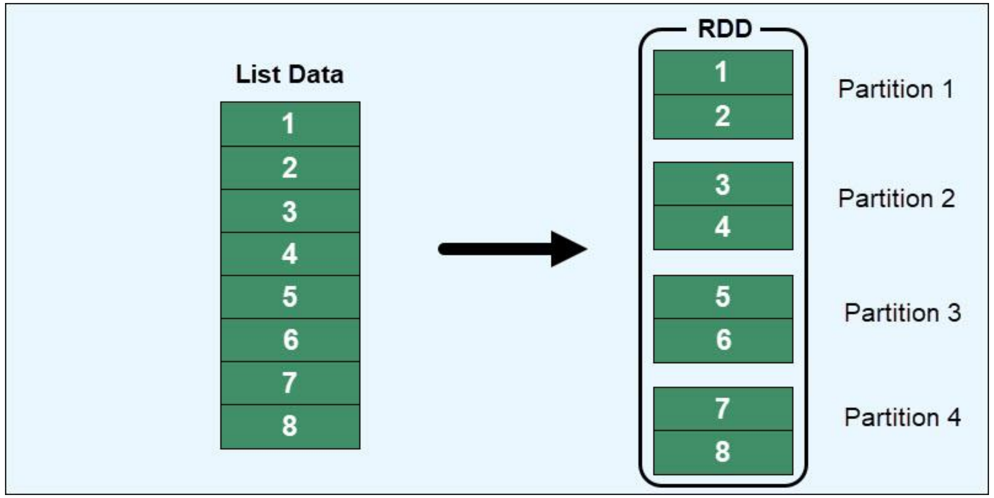
File type Pyspark can read by default : CSV , Parquet , JSON , Text

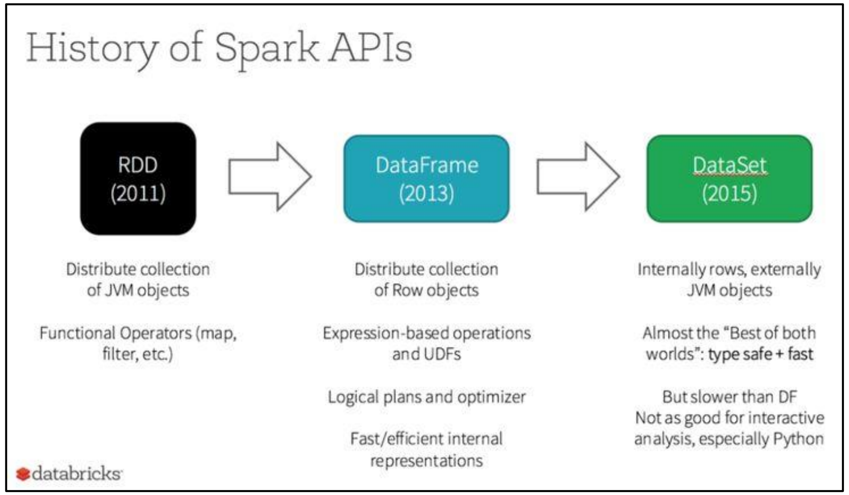
3.1 Resilient Distributed Datasets (RDD)

RDD was the primary user-facing API in Spark since its inception. At the core, **an RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers transformations and actions**.

RDDs reside in RAM through a caching process. Data that does not fit is either recalculated to reduce the size or stored on a permanent storage. Caching allows retrieving data without reading from disk, reducing disk overhead.

RDDs further distribute the data storage across multiple partitions. Partitioning allows data recovery in case a node fails and ensures the data is available at all time.





History of Spark APIs

RDD (2011)

Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

DataFrame (2013)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

DataSet (2015)

Internally rows, externally JVM objects

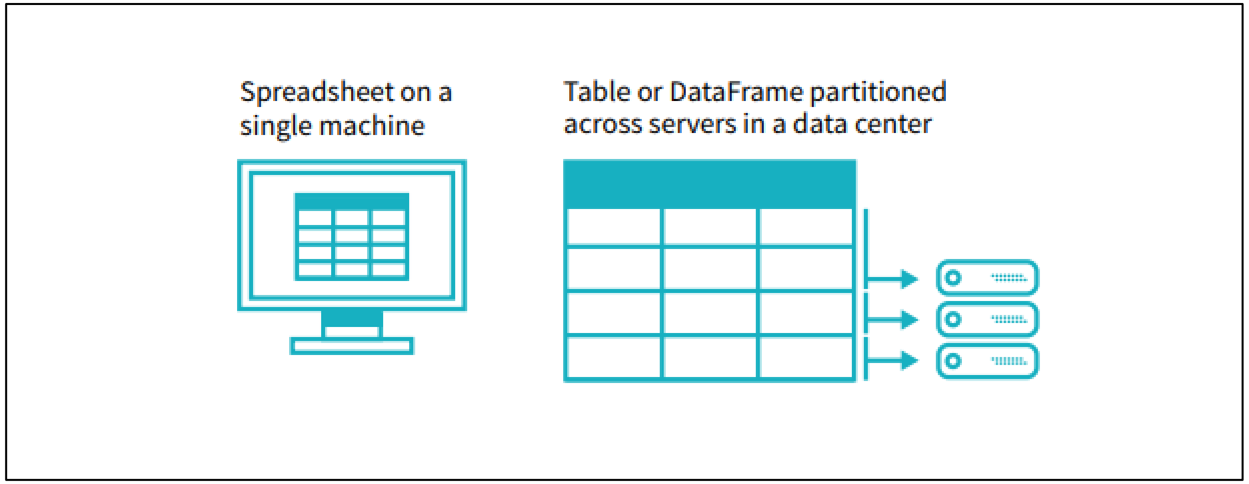
Almost the "Best of both worlds": type safe + fast

But slower than DF
Not as good for interactive analysis, especially Python

3.2 Dataframe

A DataFrame is a data structure that organizes data into a **2-dimensional table of rows and columns, much like a spreadsheet**. DataFrames are one of the most common data structures used in modern data analytics because they are a flexible and intuitive way of storing and working with data.

Every **DataFrame contains a blueprint, known as a schema, that defines the name and data type of each column.**


Spreadsheet on a single machine
Table or DataFrame partitioned across servers in a data center

3.3 Pyspark vs Pandas

Key Differences between PySpark and Pandas

1.**PySpark** is a library for working with large datasets in a distributed computing environment,
while **Pandas** is a library for working with smaller, tabular datasets on a single machine.

2.**PySpark** is built on top of the Apache Spark framework and uses the Resilient Distributed Datasets (RDD) data structure,
while **Pandas** uses the DataFrame data structure.

3.**PySpark** is designed to handle data processing tasks that are not feasible with **Pandas** due to memory constraints, such as iterative algorithms and machine learning on large datasets.

4.**PySpark** allows for parallel processing of data, while **Pandas** does not.

5.**PySpark** can read data from a variety of sources, including Hadoop Distributed File System (HDFS), Amazon S3, and local file systems,
while **Pandas** is limited to reading data from local file systems.

6.**PySpark** can be integrated with other big data tools like Hadoop and Hive, while **Pandas** is not.

7.**PySpark** is written in Scala, and runs on the Java Virtual Machine (JVM), while **Pandas** is written in Python.

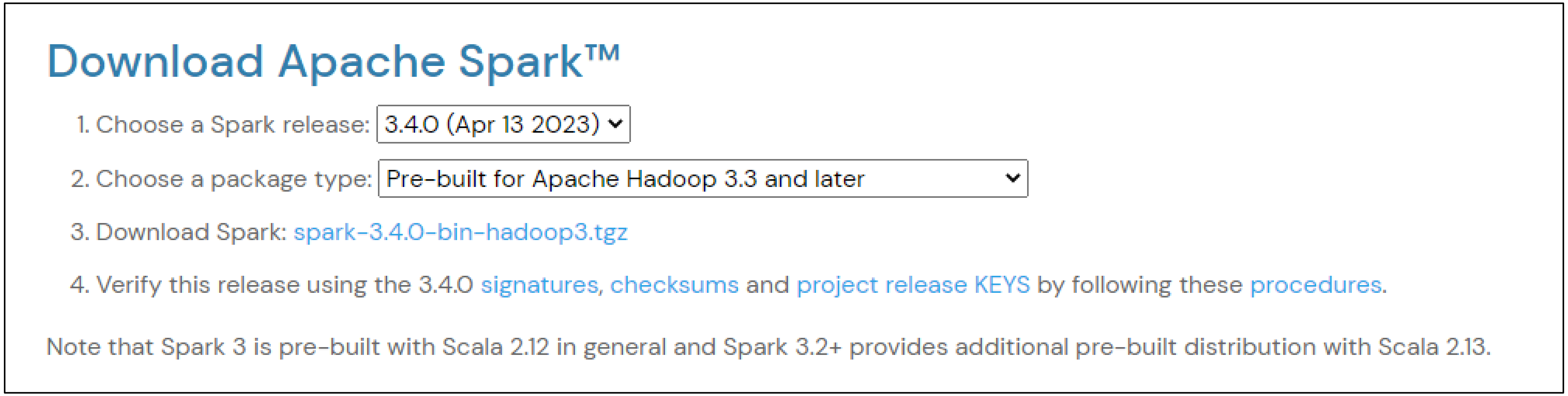
8.**PySpark** has a steeper learning curve than **Pandas**, due to the additional concepts and technologies involved (e.g. distributed computing, RDDs, Spark SQL, Spark Streaming, etc.).

4. Installing Pyspark using Python [Optional]

Pre-requisite :

- Python
- Java SDK
- Visual Studio Code

Step 1 : -Download Apache Spark
https://spark.apache.org/downloads.html



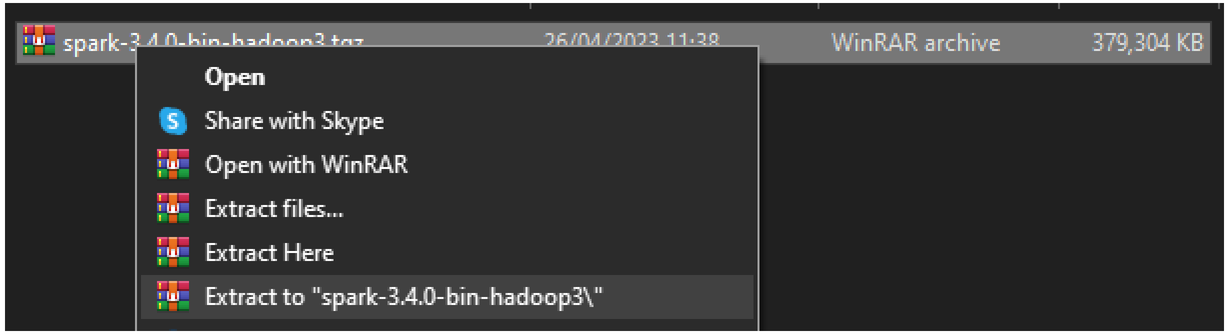
Download Apache Spark™

1. Choose a Spark release: 3.4.0 (Apr 13 2023)
2. Choose a package type: Pre-built for Apache Hadoop 3.3 and later
3. Download Spark: spark-3.4.0-bin-hadoop3.tgz
4. Verify this release using the 3.4.0 signatures, checksums and project release KEYS by following these procedures.

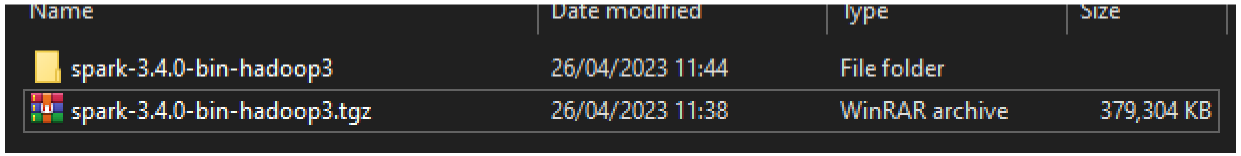
Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13.

4. Installing Pyspark using Python

Step 2 : Unzp .tgz file in any directory



spark-3.4.0-bin-hadoop3.tgz	26/04/2023 11:38	WinRAR archive	379,304 KB

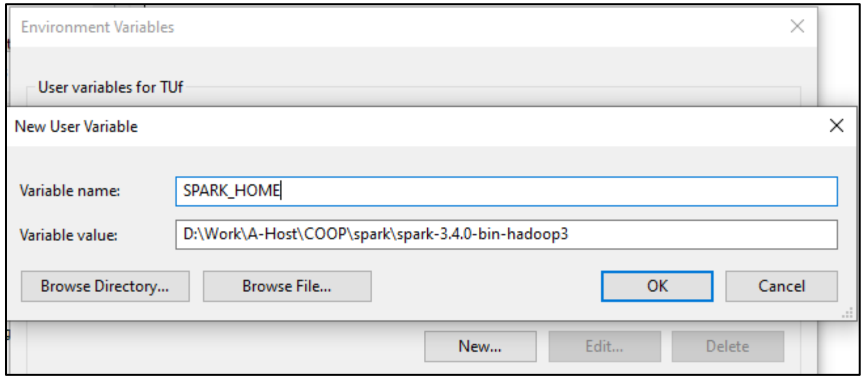


Name	Date modified	Type	Size
spark-3.4.0-bin-hadoop3	26/04/2023 11:44	File folder	
spark-3.4.0-bin-hadoop3.tgz	26/04/2023 11:38	WinRAR archive	379,304 KB

4. Installing Pyspark using Python

Step 3 : Add Environment Variable for Spark

SPARK_HOME = c:\your\home\directory\spark-3.2.1-bin-hadoop3.2
PATH = %PATH%;%SPARK_HOME%\bin



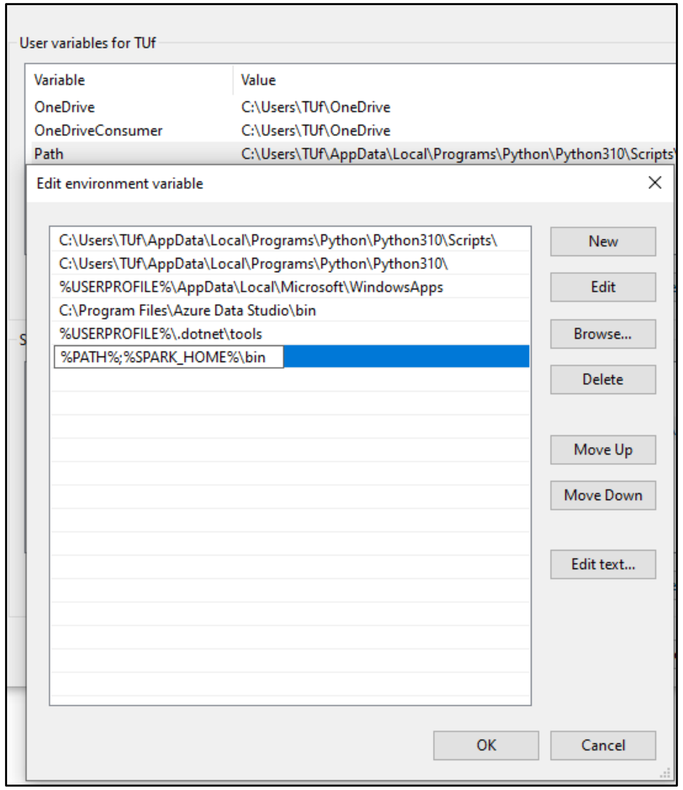
Environment Variables

User variables for TUf

New User Variable

Variable name: SPARK_HOME

Variable value: D:\Work\A-Host\COOP\spark\spark-3.4.0-bin-hadoop3



User variables for TUf

Variable	Value
OneDrive	C:\Users\TUf\OneDrive
OneDriveConsumer	C:\Users\TUf\OneDrive
Path	C:\Users\TUf\AppData\Local\Programs\Python\Python310\Scripts\

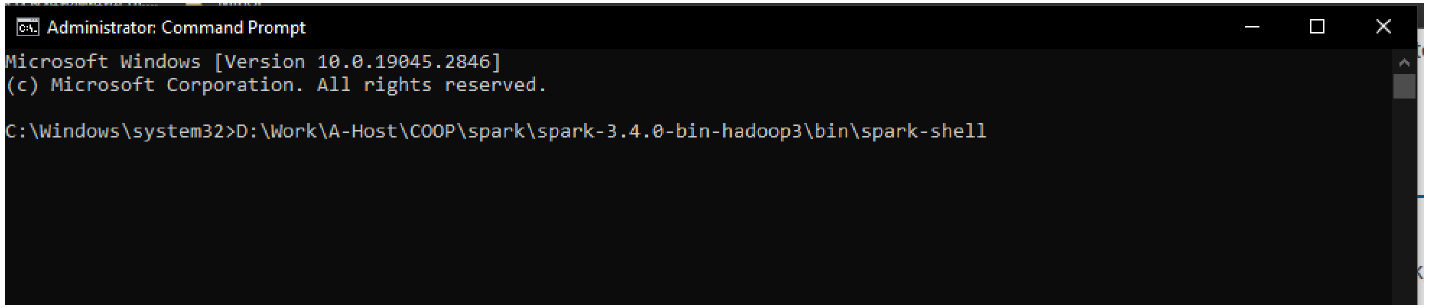
Edit environment variable

```
C:\Users\TUf\AppData\Local\Programs\Python\Python310\Scripts\
C:\Users\TUf\AppData\Local\Programs\Python\Python310\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Program Files\Azure Data Studio\bin
%USERPROFILE%\.dotnet\tools
%PATH%;%SPARK_HOME%\bin
```

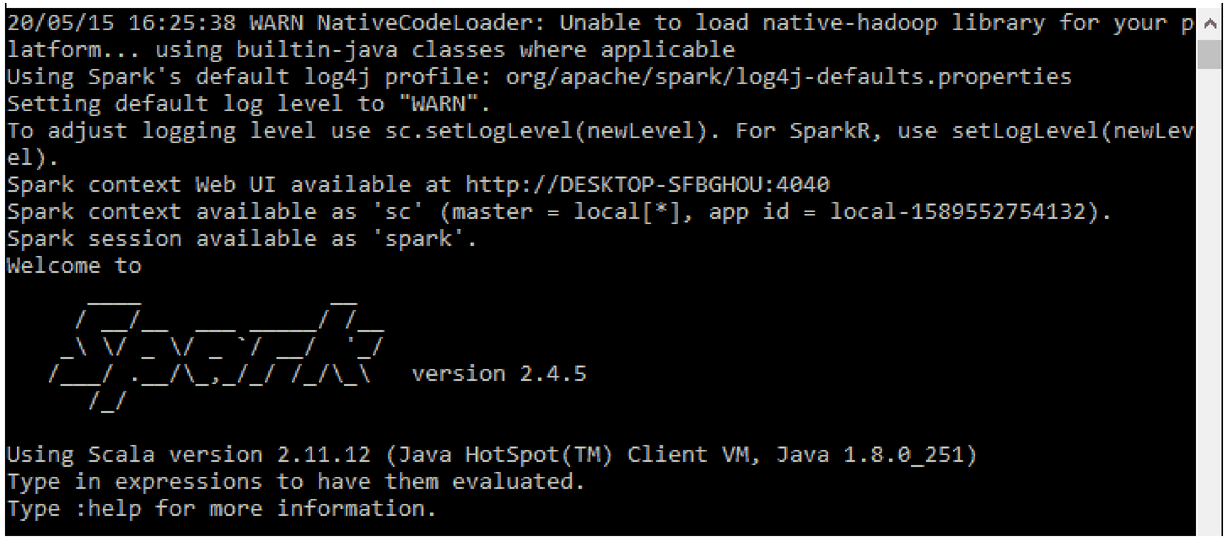
4. Installing Pyspark using Python

Step 4 : Test Run spark in command prompt as Administrator

:[Directory]\your\home\directory\spark-3.4.0-bin-hadoop3 [Version]\bin\spark-shell



```
Microsoft Windows [Version 10.0.19045.2846]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>D:\Work\A-Host\COOP\spark\spark-3.4.0-bin-hadoop3\bin\spark-shell
```

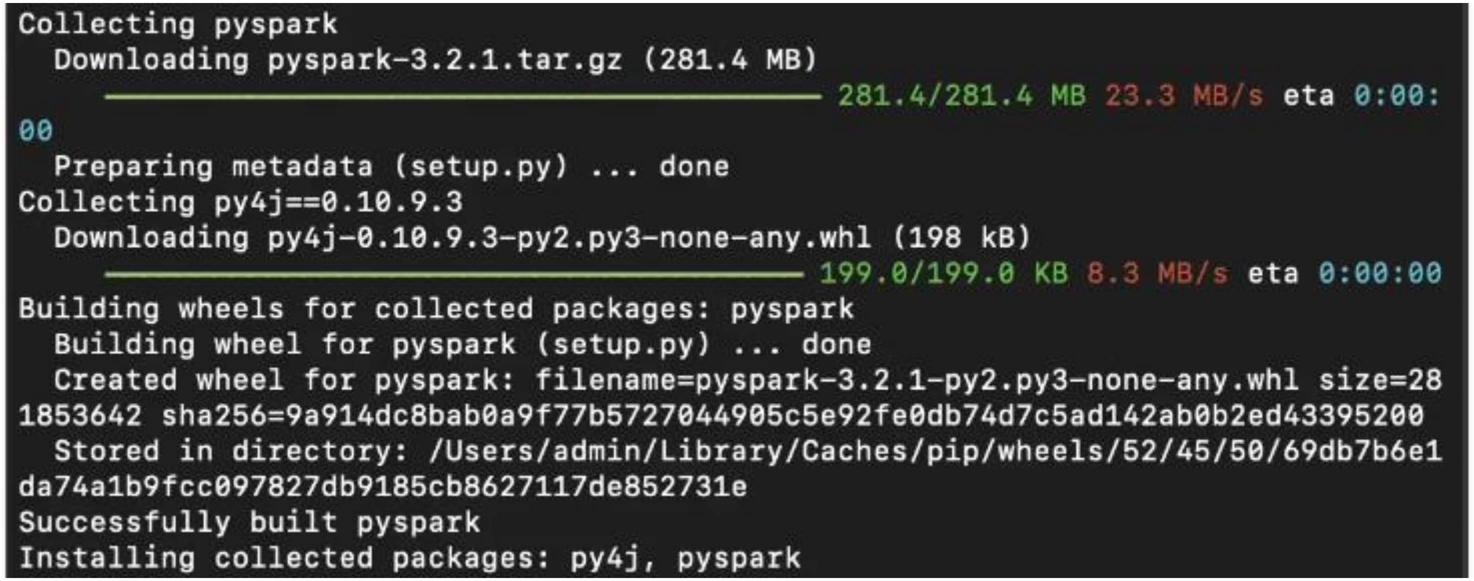


```
20/05/15 16:25:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your p
latform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLev
el).
Spark context Web UI available at http://DESKTOP-SFBGHOU:4040
Spark context available as 'sc' (master = local[*], app id = local-1589552754132).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) Client VM, Java 1.8.0_251)
Type in expressions to have them evaluated.
Type :help for more information.
```

4. Installing Pyspark using Python

Step 5 : Install Pyspark with pip

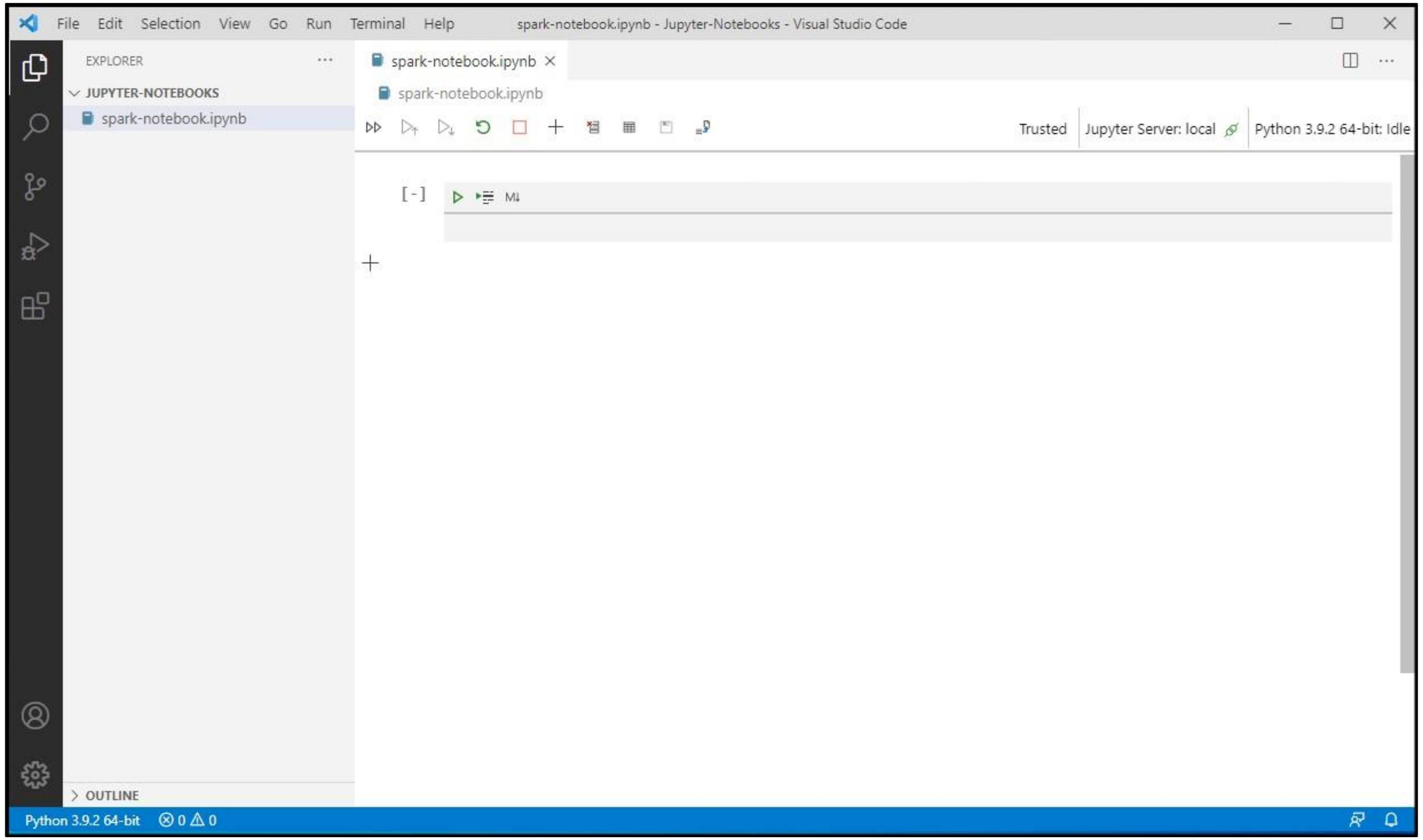
pip install pyspark



```
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 23.3 MB/s eta 0:00:
00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.0/199.0 KB 8.3 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.1-py2.py3-none-any.whl size=28
1853642 sha256=9a914dc8bab0a9f77b5727044905c5e92fe0db74d7c5ad142ab0b2ed43395200
  Stored in directory: /Users/admin/Library/Caches/pip/wheels/52/45/50/69db7b6e1
da74a1b9fcc097827db9185cb8627117de852731e
Successfully built pyspark
Installing collected packages: py4j, pyspark
```

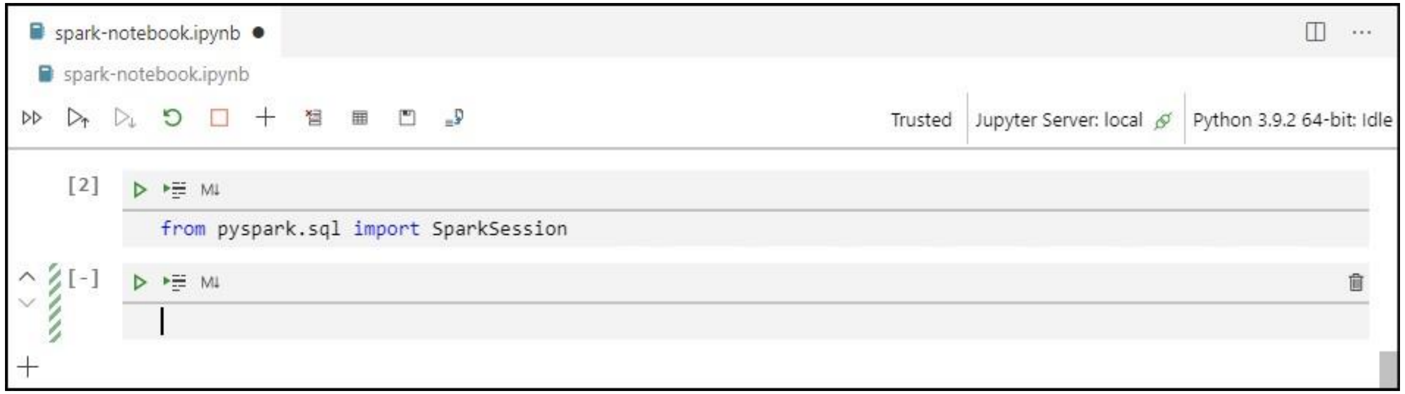
4. Installing Pyspark using Python

Step 6 : Open Visual Studio Code and Create Jupytor Notebook

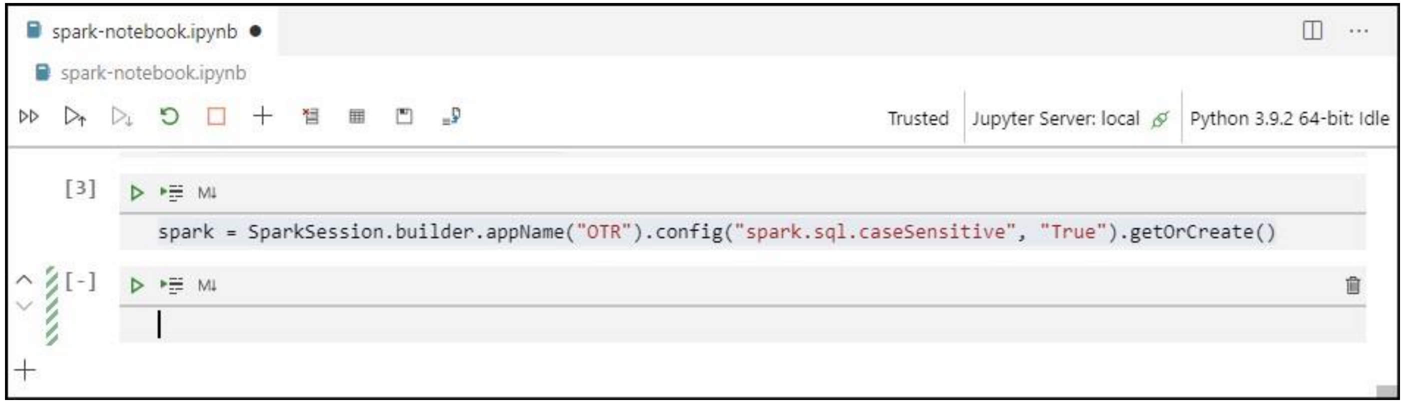

File Edit Selection View Go Run Terminal Help
spark-notebook.ipynb - Jupyter-Notebooks - Visual Studio Code
EXPLORER
JUPYTER-NOTEBOOKS
spark-notebook.ipynb
spark-notebook.ipynb
Trusted
Jupyter Server: local
Python 3.9.2 64-bit: Idle
[-]
M↓
OUTLINE
Python 3.9.2 64-bit
⊗ 0 ⚠ 0

4. Installing Pyspark using Python

Step 7 : Import Spark Session Class



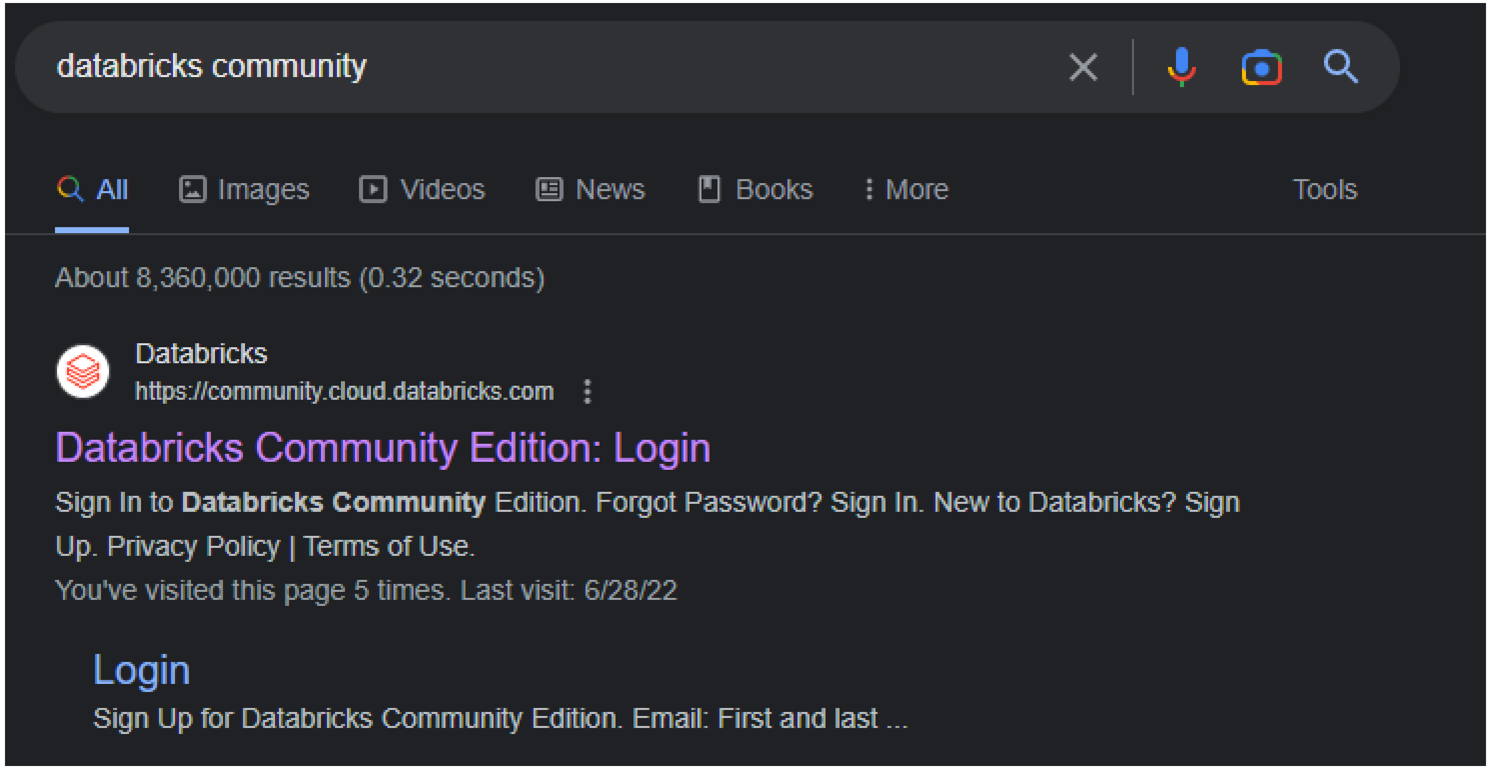
```
from pyspark.sql import SparkSession
```



```
spark = SparkSession.builder.appName("OTR").config("spark.sql.caseSensitive", "True").getOrCreate()
```

5. Register Databricks Community Edition

Step 1 :
Search ‘ **Databricks Community** ’ in Google
Or type : https://community.cloud.databricks.com/login.html



About 8,360,000 results (0.32 seconds)

Databricks
https://community.cloud.databricks.com

Databricks Community Edition: Login

Sign In to **Databricks Community** Edition. Forgot Password? Sign In. New to Databricks? Sign Up. Privacy Policy | Terms of Use.

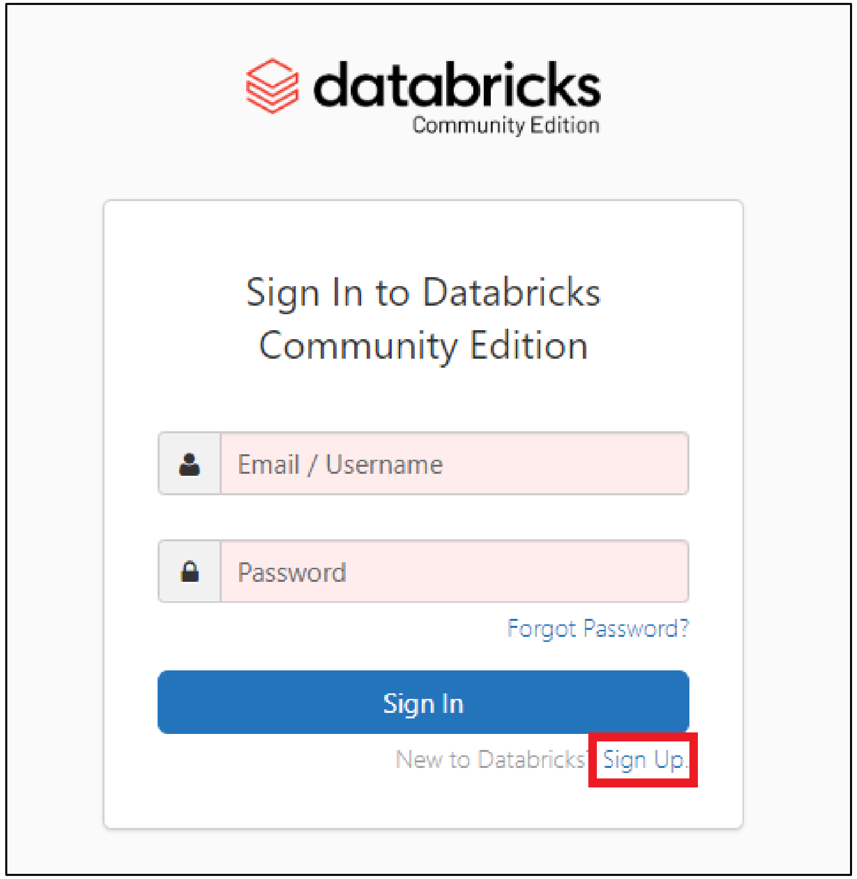
You've visited this page 5 times. Last visit: 6/28/22

Login

Sign Up for Databricks Community Edition. Email: First and last ...

5. Register Databricks Community Edition

Step 2 :
Click : **Sign Up** at the bottom left , Databricks will move to Registration page
Then fill the textbox for Creating Accounts



databricks
Community Edition

Sign In to Databricks Community Edition

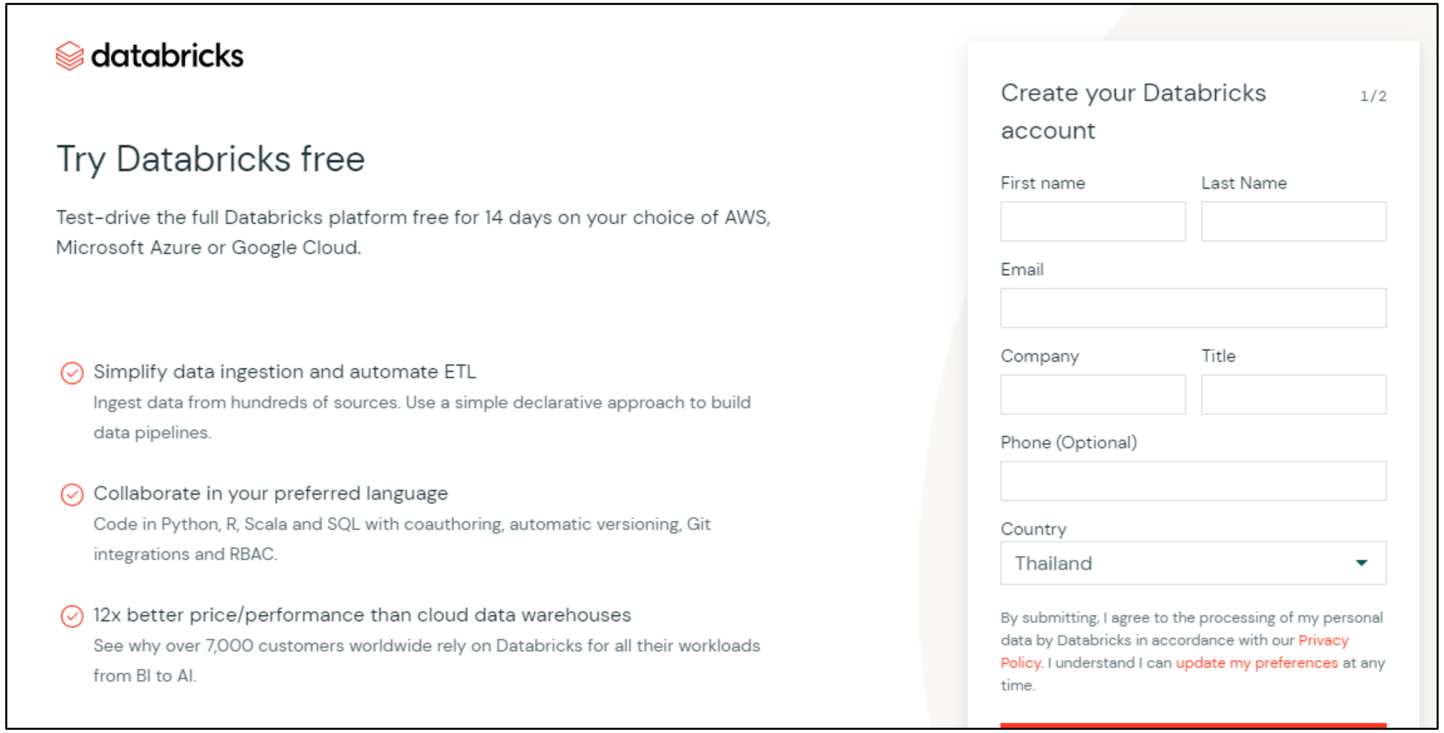
Email / Username

Password

Forgot Password?

Sign In

New to Databricks? Sign Up.



databricks

Try Databricks free

Test-drive the full Databricks platform free for 14 days on your choice of AWS, Microsoft Azure or Google Cloud.

- Simplify data ingestion and automate ETL
  Ingest data from hundreds of sources. Use a simple declarative approach to build data pipelines.
- Collaborate in your preferred language
  Code in Python, R, Scala and SQL with coauthoring, automatic versioning, Git integrations and RBAC.
- 12x better price/performance than cloud data warehouses
  See why over 7,000 customers worldwide rely on Databricks for all their workloads from BI to AI.

Create your Databricks account 1/2

First name | Last Name

Email

Company | Title

Phone (Optional)

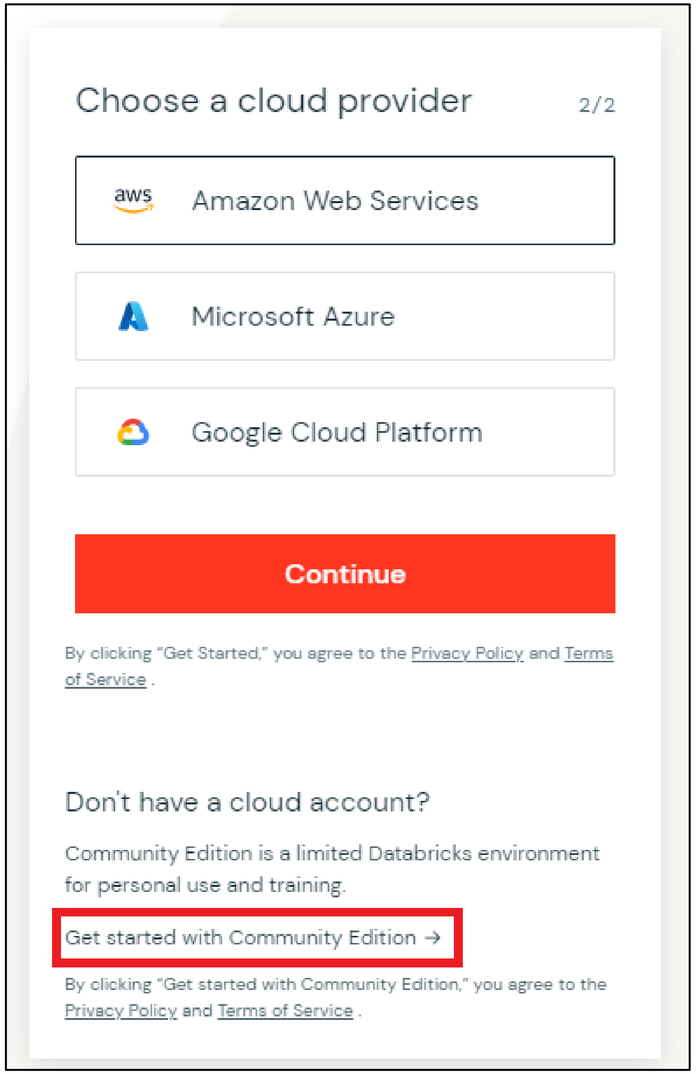
Country
Thailand

By submitting, I agree to the processing of my personal data by Databricks in accordance with our Privacy Policy. I understand I can update my preferences at any time.

5. Register Databricks Community Edition

Step 3 :

After Filled the Information and Click ‘Continue’
On choosing a Cloud Provider
Click on : ‘**Get Started with Community Edition**’
Site will Popup – Verification Test
After that system will provide link to Email that use to
Create an Account



Choose a cloud provider 2/2

Amazon Web Services

Microsoft Azure

Google Cloud Platform

Continue

By clicking "Get Started," you agree to the Privacy Policy and Terms of Service .

Don't have a cloud account?

Community Edition is a limited Databricks environment for personal use and training.

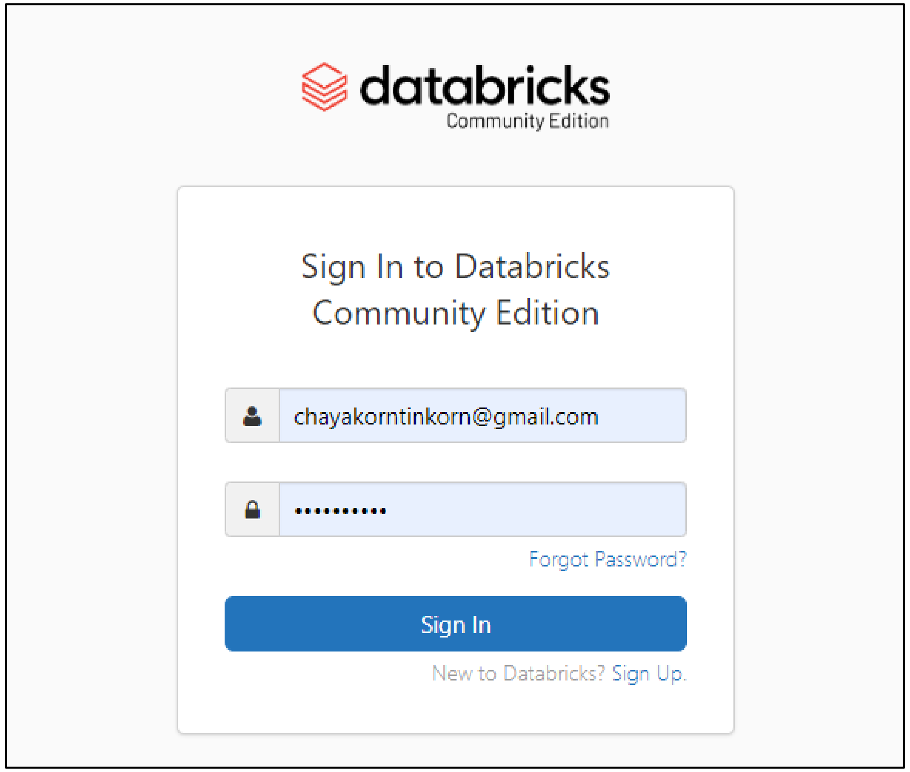
Get started with Community Edition →

By clicking "Get started with Community Edition," you agree to the Privacy Policy and Terms of Service .

5. Register Databricks Community Edition

Step 4 :

After Finish Register an Account , Back to login page then Sign In with account you created.



Sign In to Databricks Community Edition

chayakorntinkorn@gmail.com

••••••••••

Basic Python

6. Basic Python

What is Python

Python is a high-level, object-oriented programming language used in coding.
It is used for web development (server-side), software development, mathematics, system scripting.

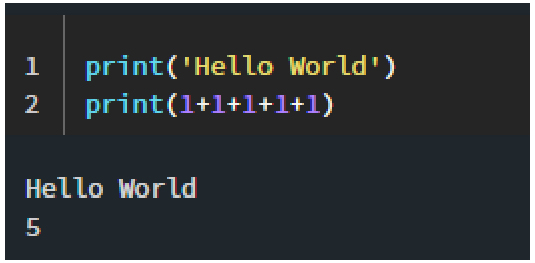
Python Syntax compared to other programming languages

•Python was designed for readability and has some similarities to the English language with influence from mathematics.
•Python **uses new lines to complete a command**, as opposed to **other programming languages which often use semicolons or parentheses**.
•Python **relies on indentation, using whitespace, to define scope**; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

6. Basic Python

Syntax

In Python Language , Python syntax can be executed by writing directly without semi-colon (;) like other Language.



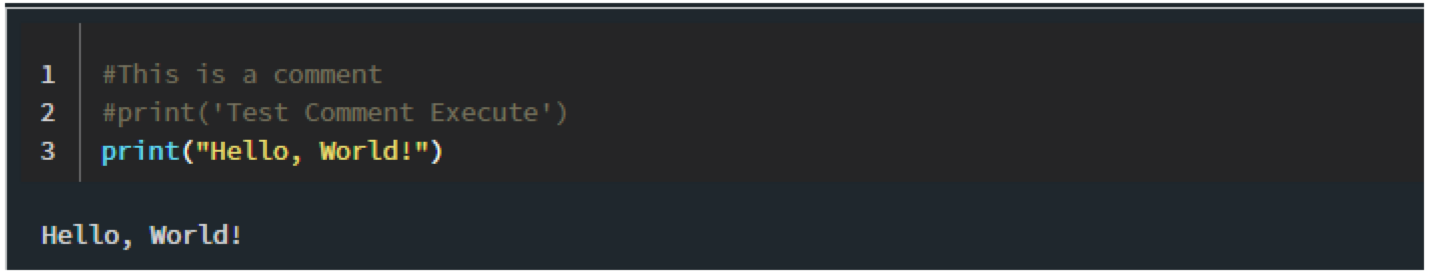
```
print('Hello World')
print(1+1+1+1+1)
```

```
Hello World
5
```

6. Basic Python

Comments

Comments starts with a ‘ **#** ’ and Python will ignore them
prevent execution when testing code or Explain Python Code



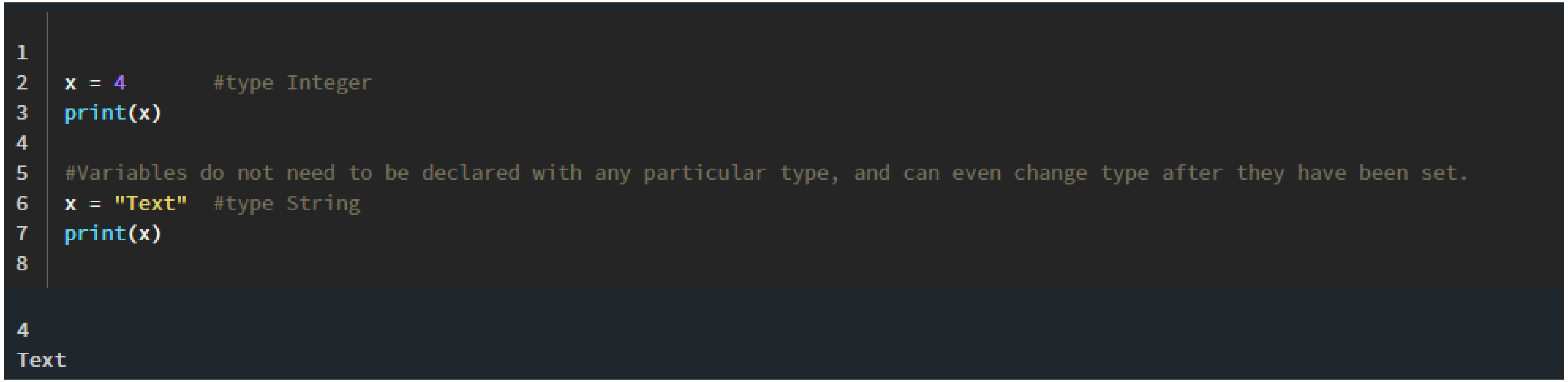
```
#This is a comment
#print('Test Comment Execute')
print("Hello, World!")
```

```
Hello, World!
```

6. Basic Python

Variables

Variables are containers for storing data values.
Python has no command for declaring a variable.
A variable is created the moment the values assigned to it.



```

x = 4       #type Integer
print(x)

#Variables do not need to be declared with any particular type, and can even change type after they have been set.
x = "Text"  #type String
print(x)

```

```
4
Text
```

6. Basic Python

Casting (Change Variable type)

Casting in python is therefore done using constructor functions:
int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

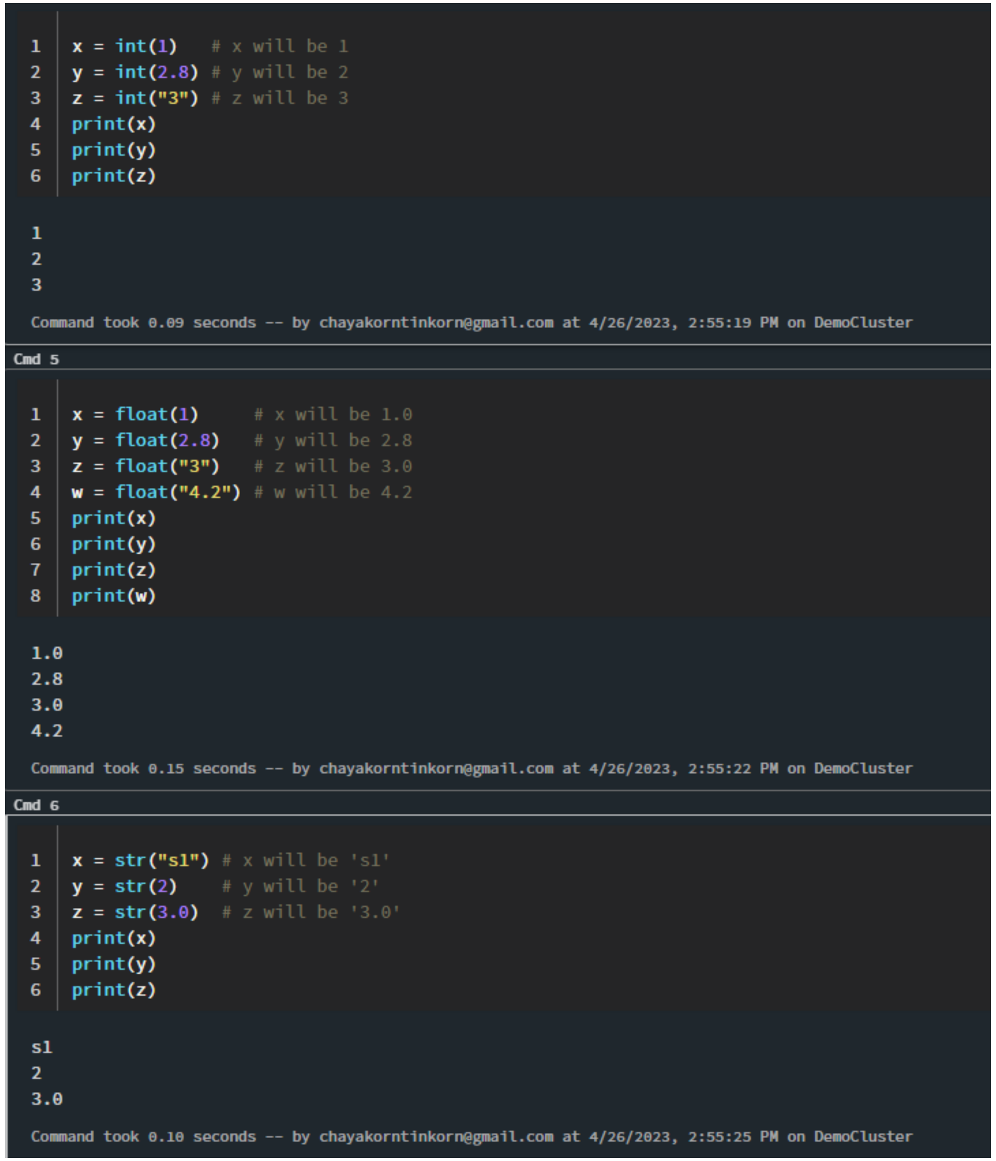
6. Basic Python

Casting

int()

float()

str()



```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
print(x)
print(y)
print(z)
```

```
1
2
3
```

Command took 0.09 seconds -- by chayakorntinkorn@gmail.com at 4/26/2023, 2:55:19 PM on DemoCluster

Cmd 5

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
print(x)
print(y)
print(z)
print(w)
```

```
1.0
2.8
3.0
4.2
```

Command took 0.15 seconds -- by chayakorntinkorn@gmail.com at 4/26/2023, 2:55:22 PM on DemoCluster

Cmd 6

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
print(x)
print(y)
print(z)
```

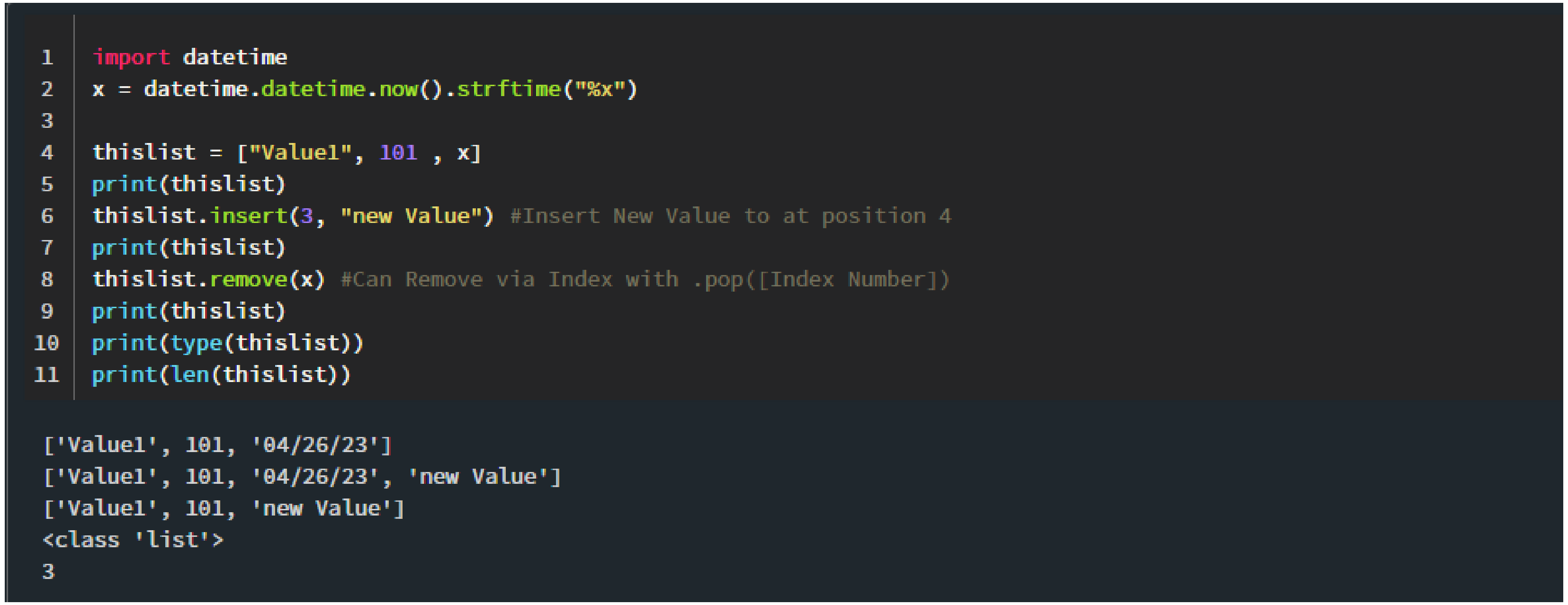
```
s1
2
3.0
```

Command took 0.10 seconds -- by chayakorntinkorn@gmail.com at 4/26/2023, 2:55:25 PM on DemoCluster

6. Basic Python

List

Lists are used to **store multiple items in a single variable**.
Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
Lists are created using **square brackets**: **[ ]**
The list is **changeable**, meaning that we can change, add, and remove items in a list after it has been created.



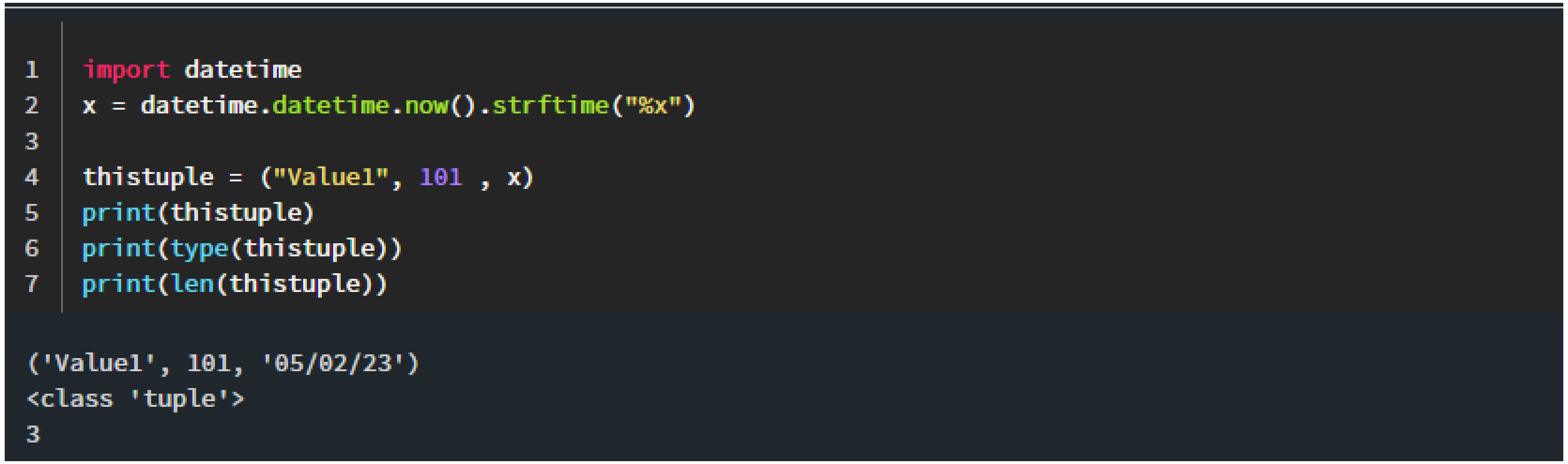
```
import datetime
x = datetime.datetime.now().strftime("%x")

thislist = ["Value1", 101 , x]
print(thislist)
thislist.insert(3, "new Value") #Insert New Value to at position 4
print(thislist)
thislist.remove(x) #Can Remove via Index with .pop([Index Number])
print(thislist)
print(type(thislist))
print(len(thislist))
```

```
['Value1', 101, '04/26/23']
['Value1', 101, '04/26/23', 'new Value']
['Value1', 101, 'new Value']
<class 'list'>
3
```

6. Basic Python

Tuple

Tuples are used **to store multiple items in a single variable**.
Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
A tuple is a **collection which is ordered and unchangeable**.
Tuples are written with **round brackets. ( )**



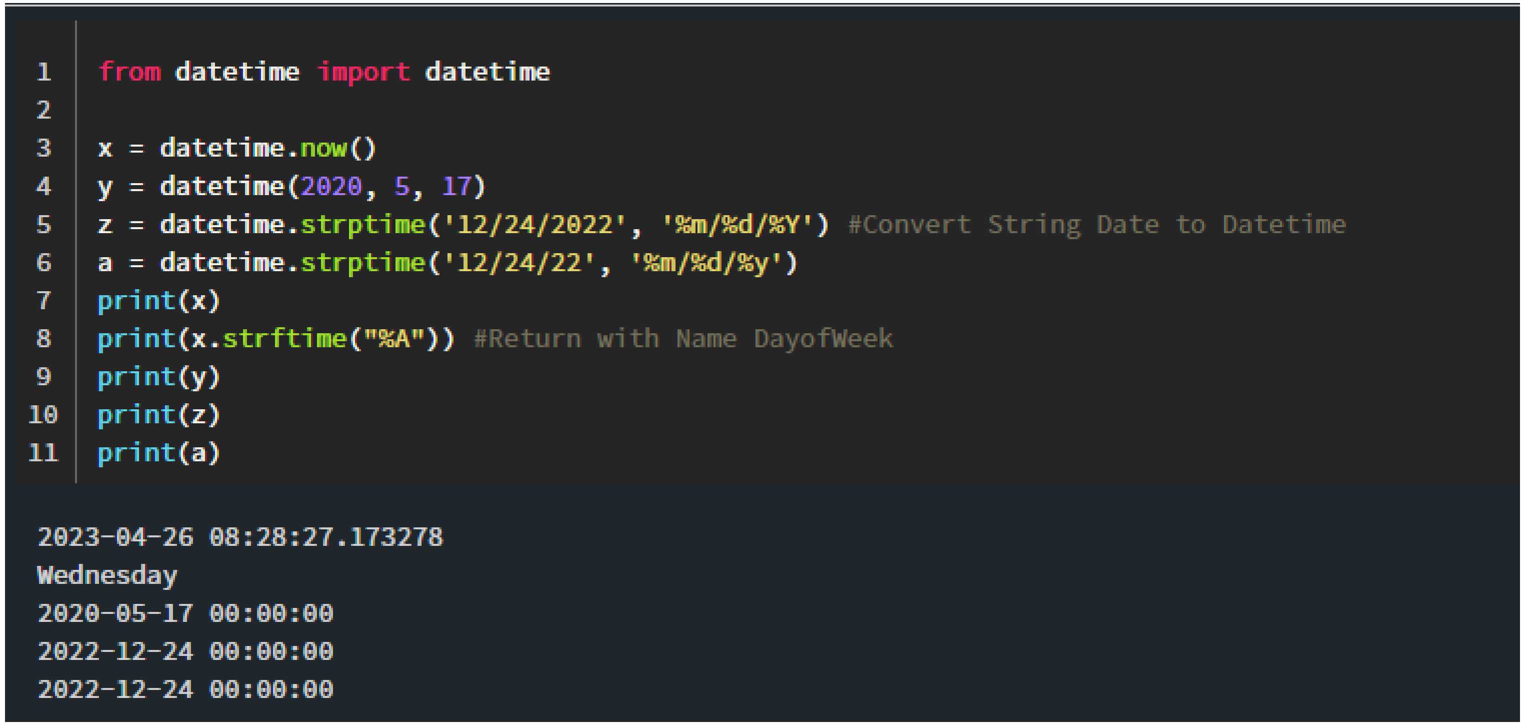
```python
import datetime
x = datetime.datetime.now().strftime("%x")

thistuple = ("Value1", 101 , x)
print(thistuple)
print(type(thistuple))
print(len(thistuple))
```

```
('Value1', 101, '05/02/23')
<class 'tuple'>
3
```

6. Basic Python

Dates

A date in Python is not a data type of its own, but we can import a module named **datetime** to work with dates as date objects.
The datetime object has a method for formatting date objects into readable strings , **strftime()** and takes one parameter, format, to **specify the format** of the returned string:



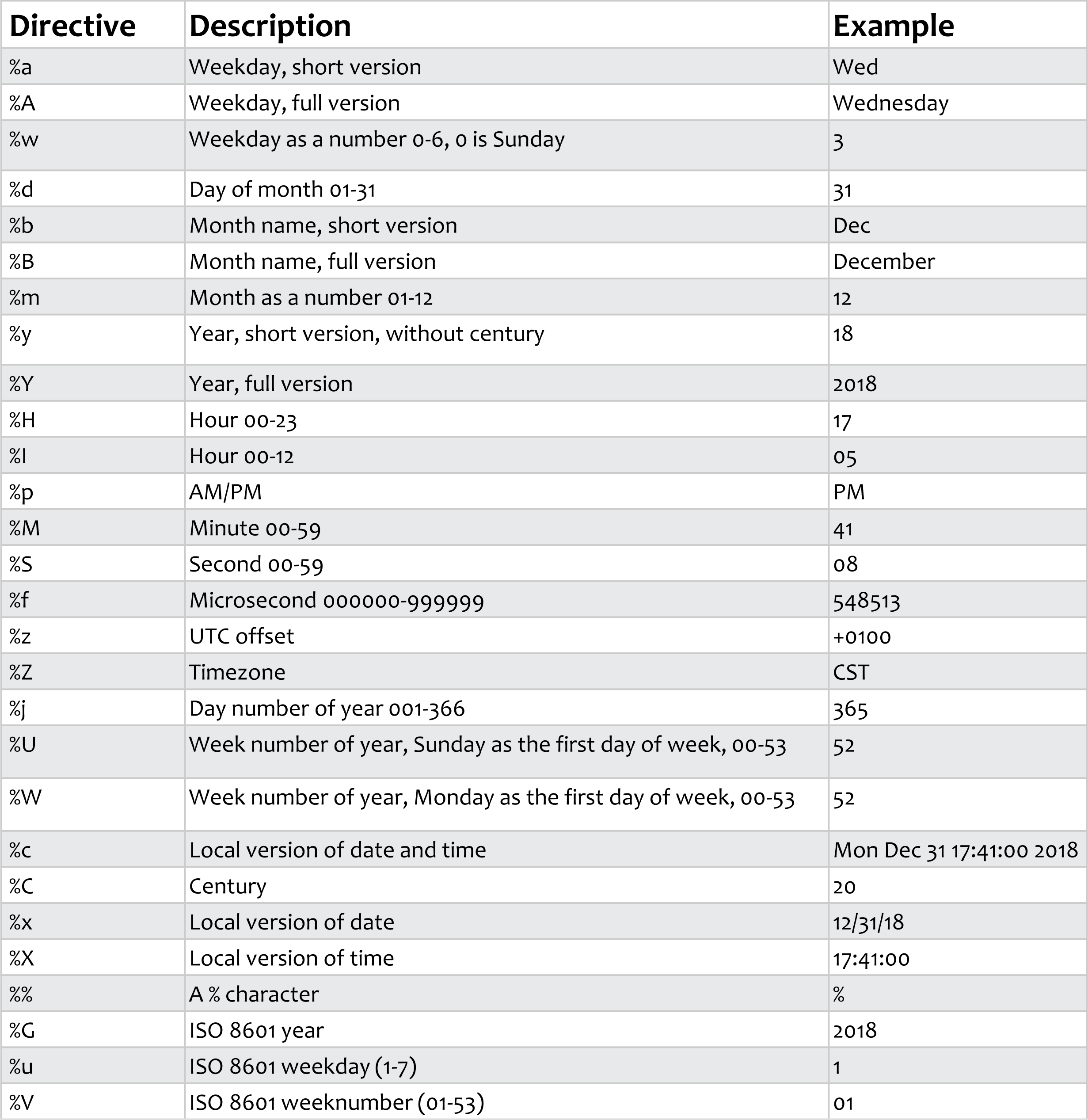
```python
from datetime import datetime

x = datetime.now()
y = datetime(2020, 5, 17)
z = datetime.strptime('12/24/2022', '%m/%d/%Y') #Convert String Date to Datetime
a = datetime.strptime('12/24/22', '%m/%d/%y')
print(x)
print(x.strftime("%A")) #Return with Name DayofWeek
print(y)
print(z)
print(a)
```

```
2023-04-26 08:28:27.173278
Wednesday
2020-05-17 00:00:00
2022-12-24 00:00:00
2022-12-24 00:00:00
```

6. Basic Python

Dates

strftime() format code

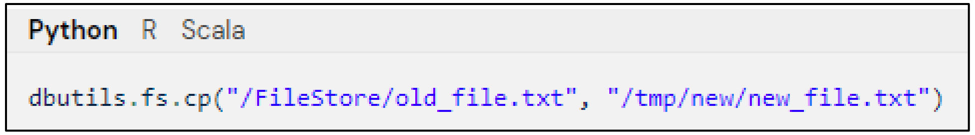


Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Databrick DBUTILS

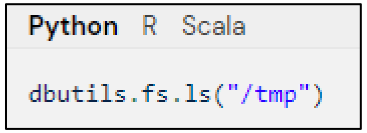
7. DATABRICKS DBUTILS COMMAND (File System)

Copy file command



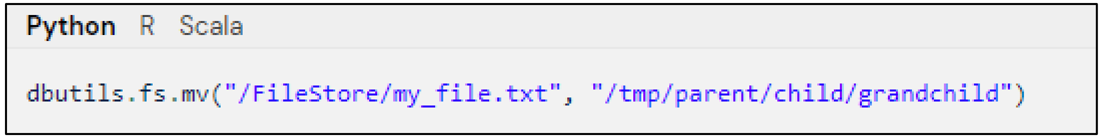
```
dbutils.fs.cp("/FileStore/old_file.txt", "/tmp/new/new_file.txt")
```

Show list of file command



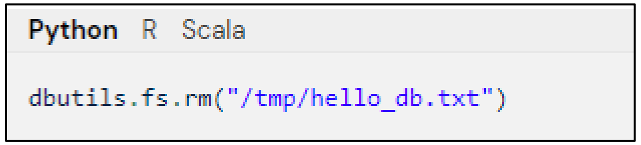
```
dbutils.fs.ls("/tmp")
```

Move file command



```
dbutils.fs.mv("/FileStore/my_file.txt", "/tmp/parent/child/grandchild")
```

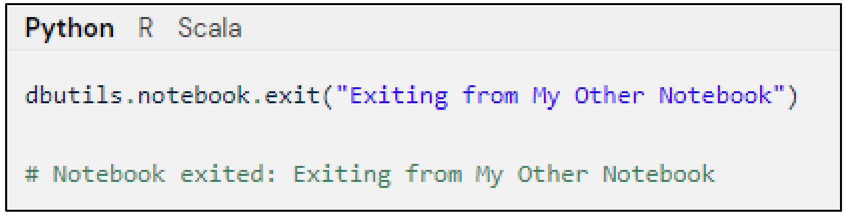
Remove file command



```
dbutils.fs.rm("/tmp/hello_db.txt")
```

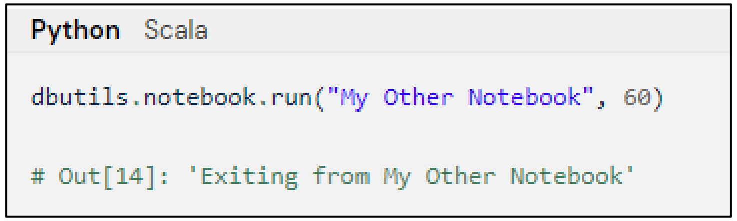
7. DATABRICKS DBUTILS COMMAND (Notebook)

Stop Execute / Exits a notebook with a value.



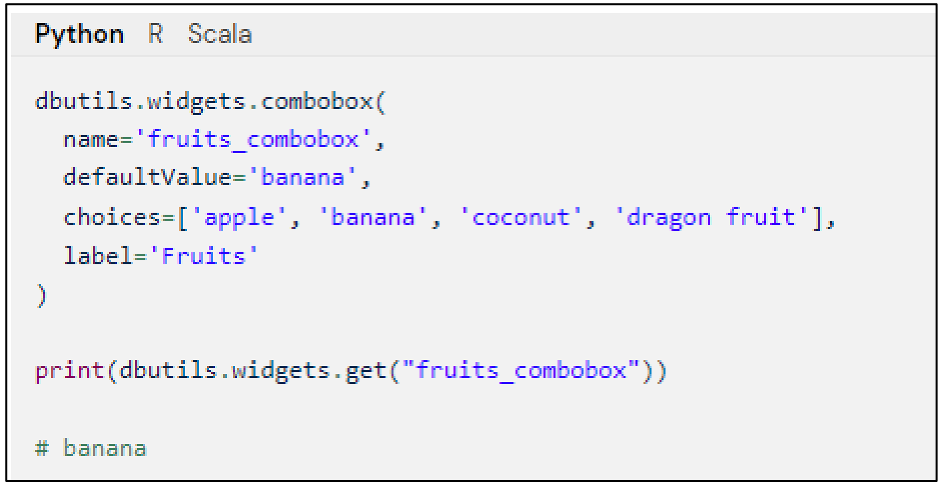
```
dbutils.notebook.exit("Exiting from My Other Notebook")

# Notebook exited: Exiting from My Other Notebook
```

Runs a notebook and returns its exit value



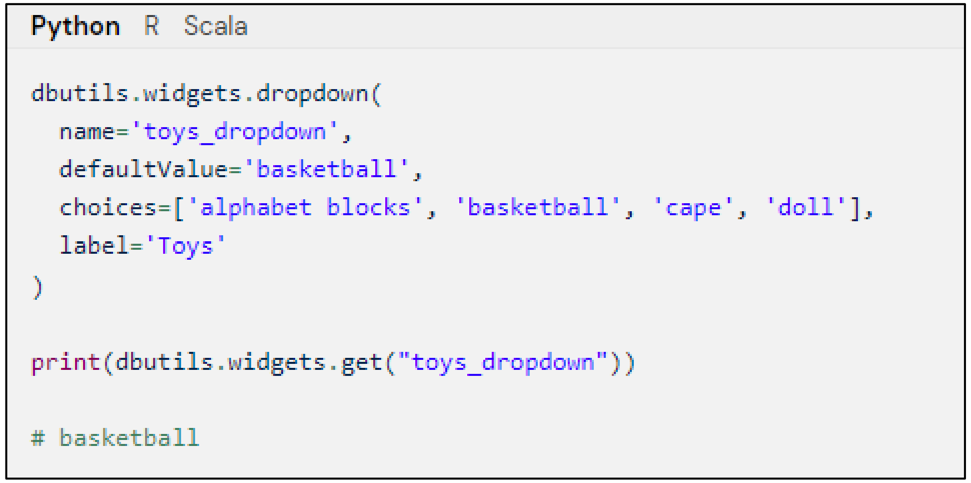
```
dbutils.notebook.run("My Other Notebook", 60)

# Out[14]: 'Exiting from My Other Notebook'
```

7. DATABRICKS DBUTILS COMMAND (Widget)



```python
dbutils.widgets.combobox(
  name='fruits_combobox',
  defaultValue='banana',
  choices=['apple', 'banana', 'coconut', 'dragon fruit'],
  label='Fruits'
)

print(dbutils.widgets.get("fruits_combobox"))

# banana
```

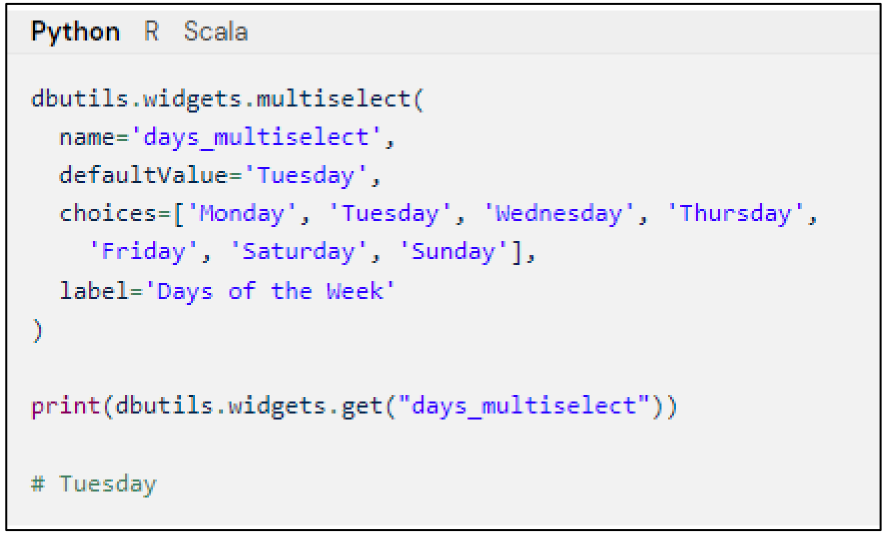
Widget ComboBox with Choice values



```python
dbutils.widgets.dropdown(
  name='toys_dropdown',
  defaultValue='basketball',
  choices=['alphabet blocks', 'basketball', 'cape', 'doll'],
  label='Toys'
)

print(dbutils.widgets.get("toys_dropdown"))

# basketball
```

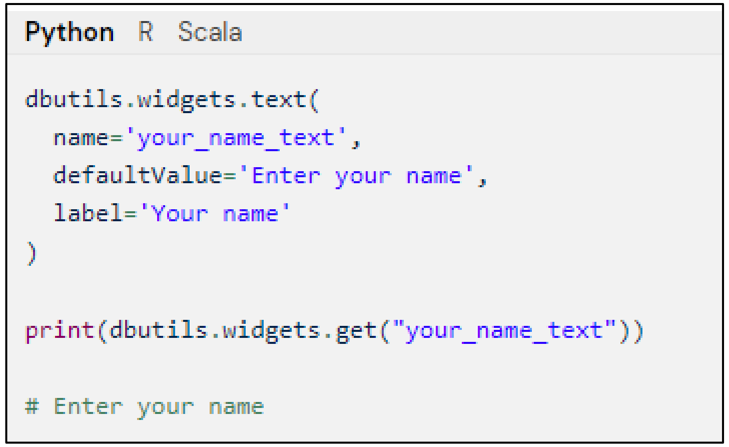
Widget Dropdown with Choice values

7. DATABRICKS DBUTILS COMMAND (Widget)



```python
dbutils.widgets.multiselect(
  name='days_multiselect',
  defaultValue='Tuesday',
  choices=['Monday', 'Tuesday', 'Wednesday', 'Thursday',
    'Friday', 'Saturday', 'Sunday'],
  label='Days of the Week'
)

print(dbutils.widgets.get("days_multiselect"))

# Tuesday
```

Widget Multi-Selection with Choice values

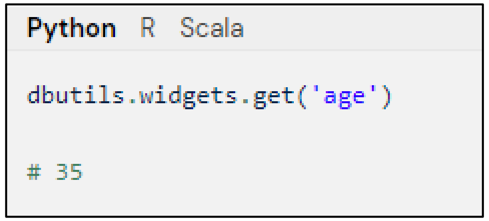


```python
dbutils.widgets.text(
  name='your_name_text',
  defaultValue='Enter your name',
  label='Your name'
)

print(dbutils.widgets.get("your_name_text"))

# Enter your name
```

Widget Textbox for Input value

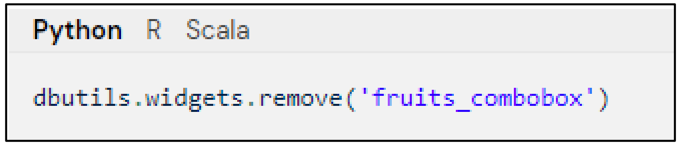
7. DATABRICKS DBUTILS COMMAND (Widget)

Get widget value



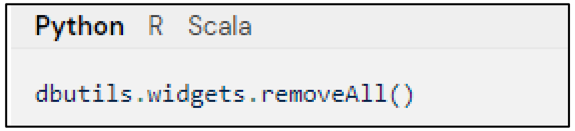
```
dbutils.widgets.get('age')

# 35
```

Remove specific widget



```
dbutils.widgets.remove('fruits_combobox')
```

Remove all exists widget



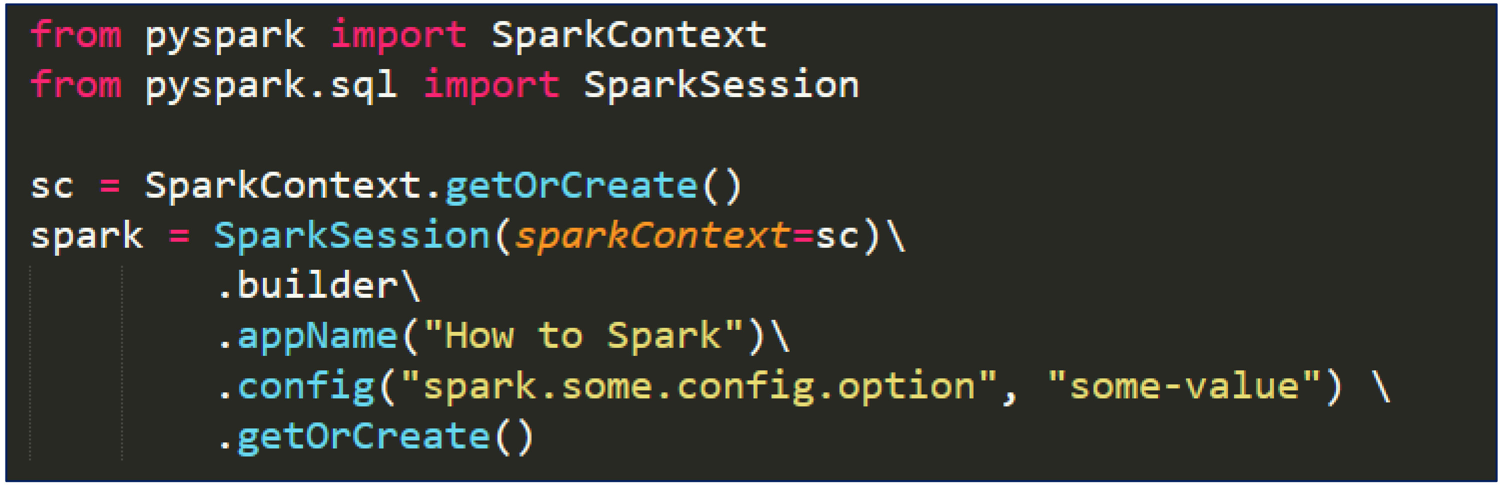
```
dbutils.widgets.removeAll()
```

Pyspark Basic

8. Pyspark Basic Command

Create Spark Session to Run Pyspark Command

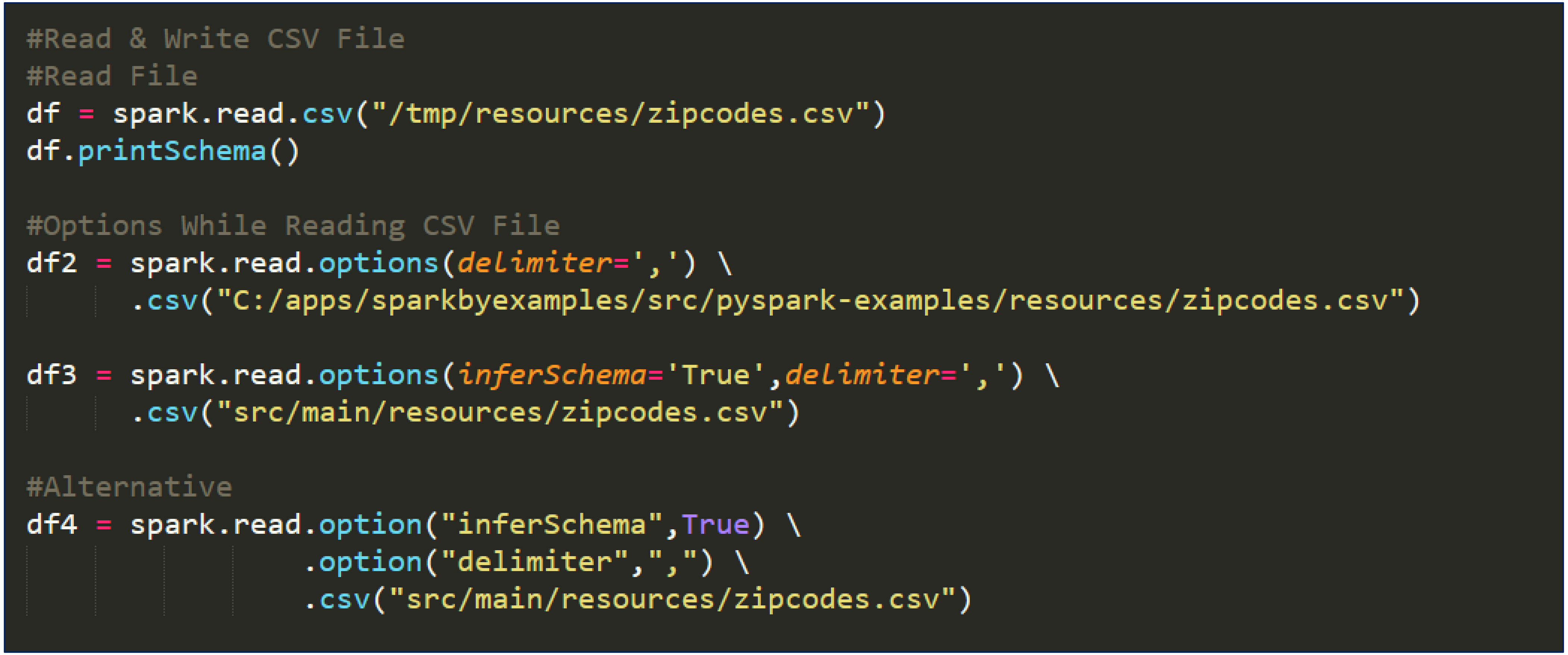
*Create in case not using on Databricks Since Databricks create Spark Session by default.



```
from pyspark import SparkContext
from pyspark.sql import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession(sparkContext=sc)\
        .builder\
        .appName("How to Spark")\
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

8. Pyspark Basic Command

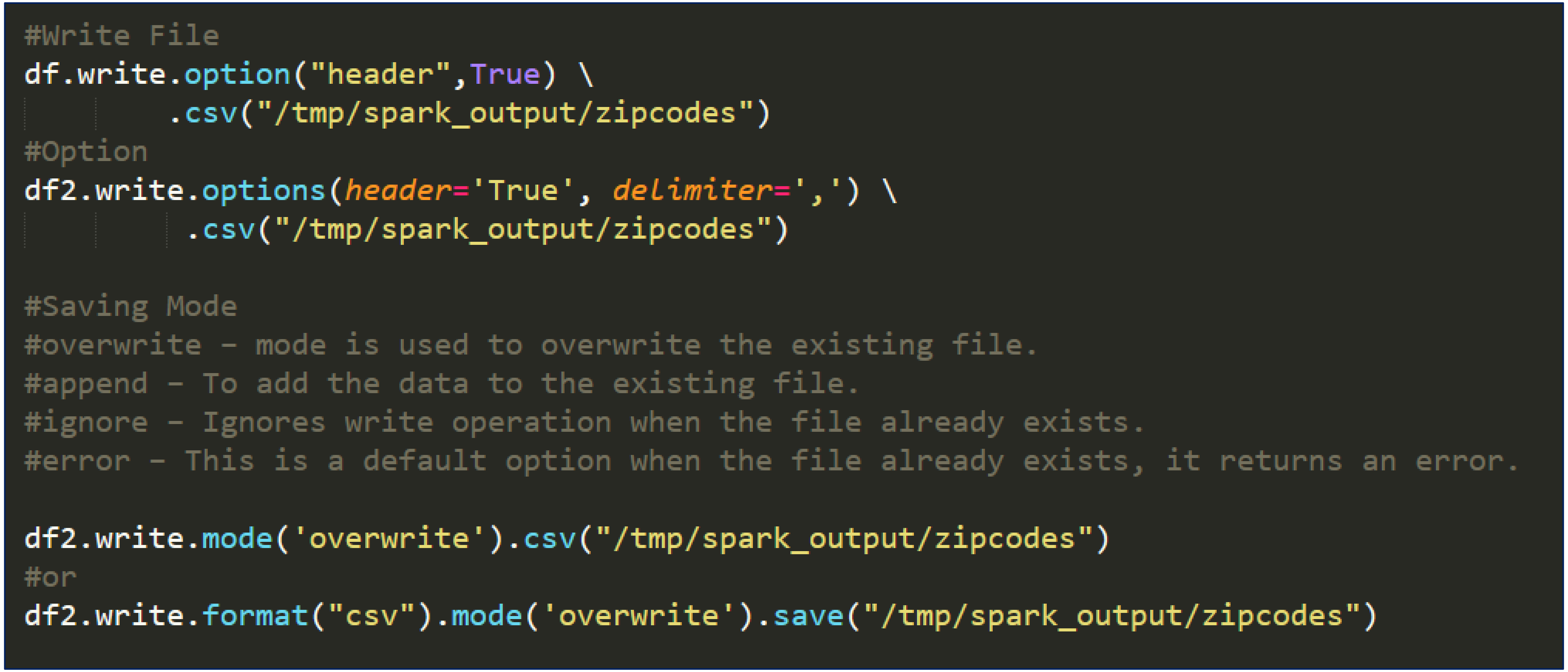
Create Dataframe from CSV file.



```
#Read & Write CSV File
#Read File
df = spark.read.csv("/tmp/resources/zipcodes.csv")
df.printSchema()

#Options While Reading CSV File
df2 = spark.read.options(delimiter=',') \
      .csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/zipcodes.csv")

df3 = spark.read.options(inferSchema='True',delimiter=',') \
      .csv("src/main/resources/zipcodes.csv")

#Alternative
df4 = spark.read.option("inferSchema",True) \
                .option("delimiter",",") \
                .csv("src/main/resources/zipcodes.csv")
```

8. Pyspark Basic Command

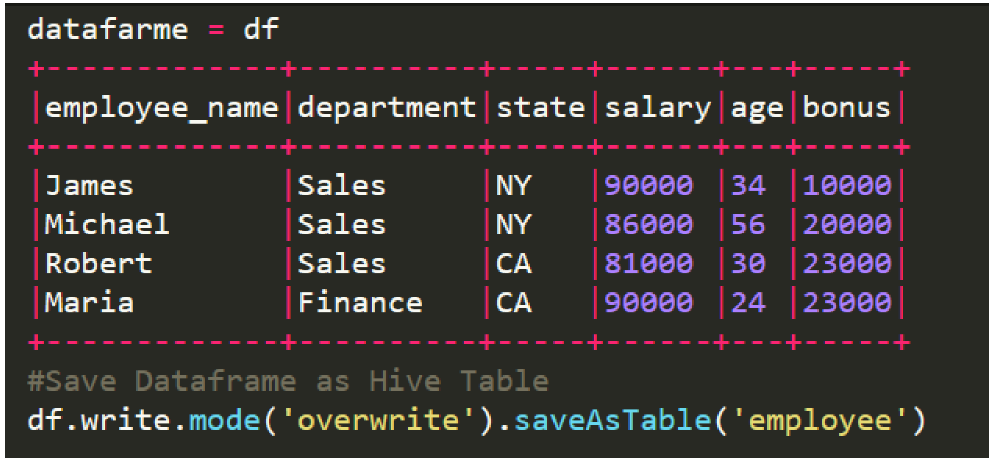
Write Dataframe into CSV file.



```
#Write File
df.write.option("header",True) \
        .csv("/tmp/spark_output/zipcodes")
#Option
df2.write.options(header='True', delimiter=',') \
         .csv("/tmp/spark_output/zipcodes")

#Saving Mode
#overwrite – mode is used to overwrite the existing file.
#append – To add the data to the existing file.
#ignore – Ignores write operation when the file already exists.
#error – This is a default option when the file already exists, it returns an error.

df2.write.mode('overwrite').csv("/tmp/spark_output/zipcodes")
#or
df2.write.format("csv").mode('overwrite').save("/tmp/spark_output/zipcodes")
```

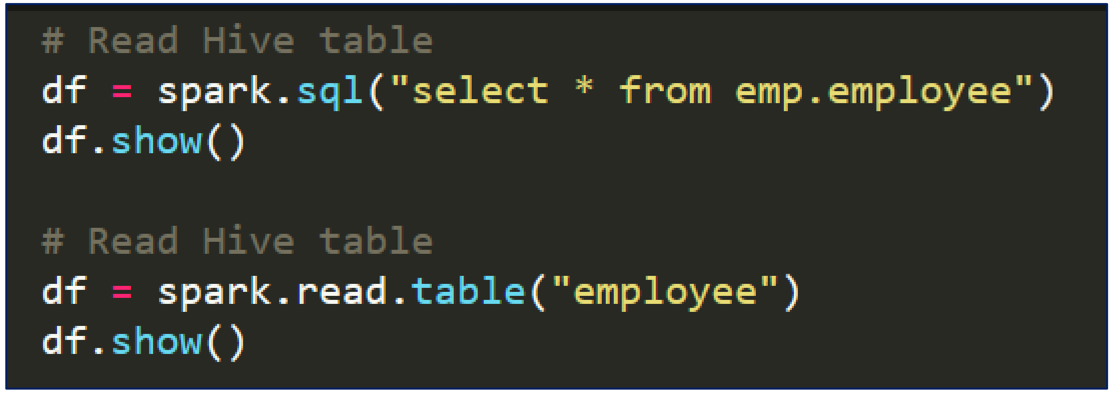
8. Pyspark Basic Command

Create Table from Dataframe in Hive.



```
datafarme = df
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Michael      |Sales     |NY   |86000 |56 |20000|
|Robert       |Sales     |CA   |81000 |30 |23000|
|Maria        |Finance   |CA   |90000 |24 |23000|
+-------------+----------+-----+------+---+-----+
#Save Dataframe as Hive Table
df.write.mode('overwrite').saveAsTable('employee')
```

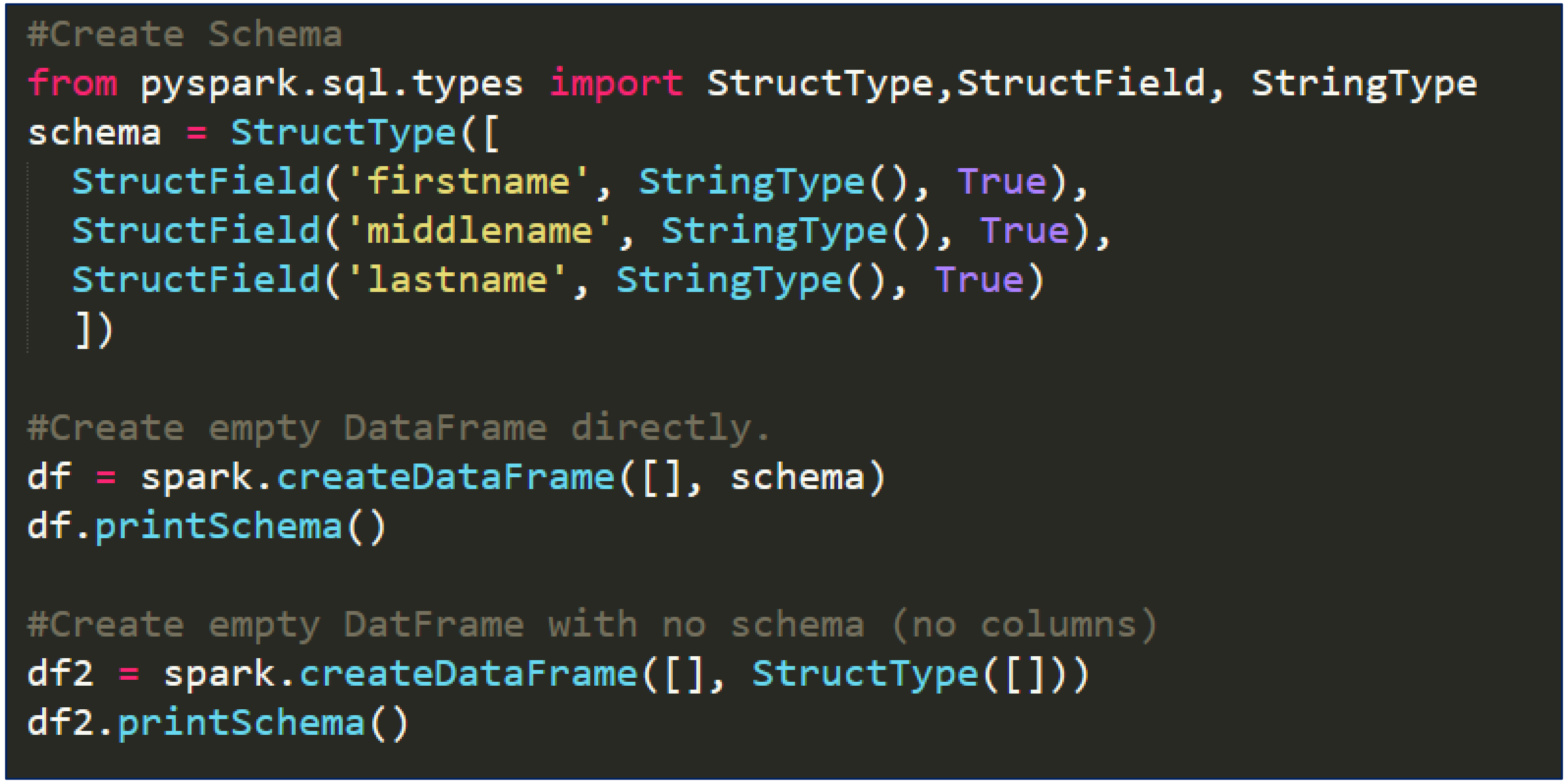
Create Dataframe from Table in Hive.



```
# Read Hive table
df = spark.sql("select * from emp.employee")
df.show()

# Read Hive table
df = spark.read.table("employee")
df.show()
```

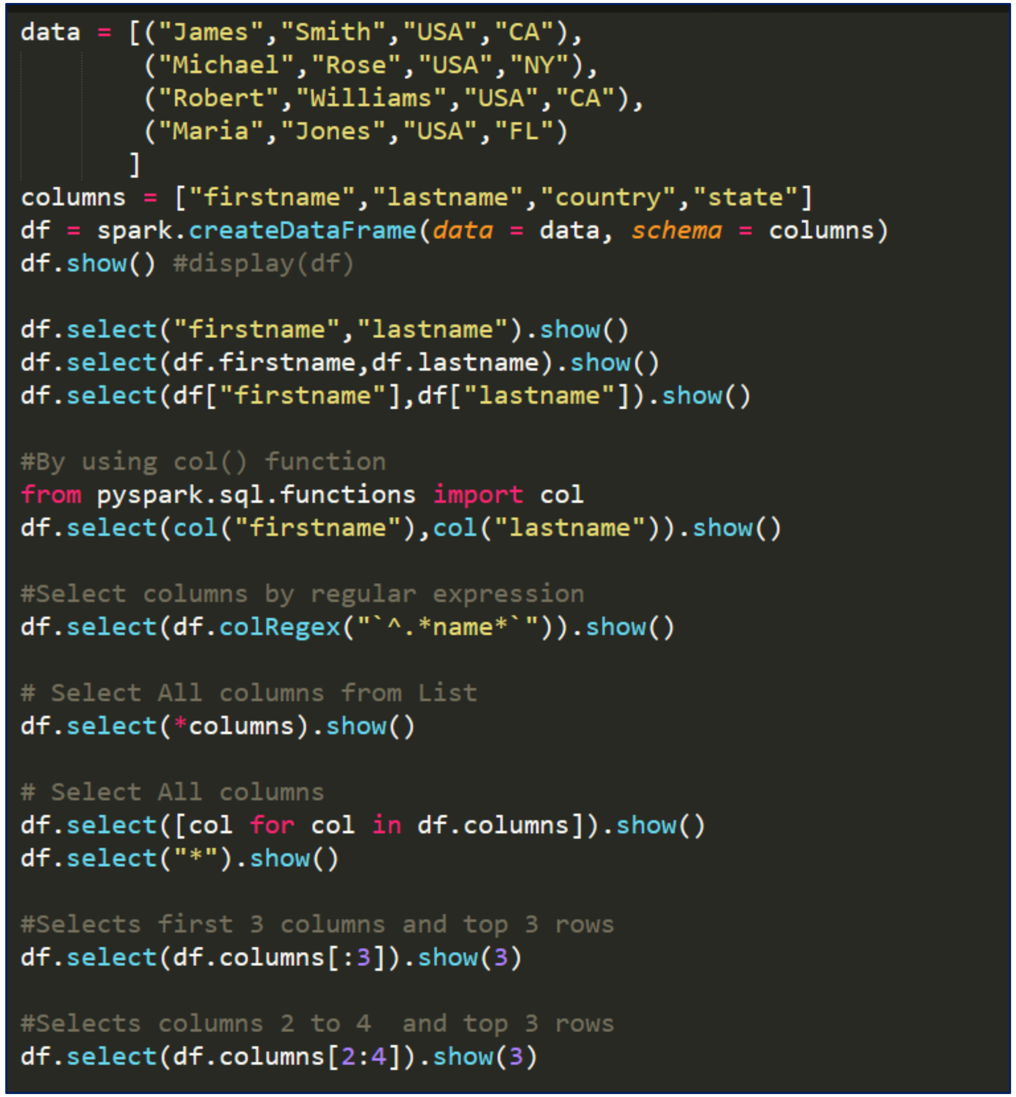
8. Pyspark Basic Command

Create New Dataframe



```
#Create Schema
from pyspark.sql.types import StructType,StructField, StringType
schema = StructType([
  StructField('firstname', StringType(), True),
  StructField('middlename', StringType(), True),
  StructField('lastname', StringType(), True)
  ])

#Create empty DataFrame directly.
df = spark.createDataFrame([], schema)
df.printSchema()

#Create empty DatFrame with no schema (no columns)
df2 = spark.createDataFrame([], StructType([]))
df2.printSchema()
```

8. Pyspark Basic Command

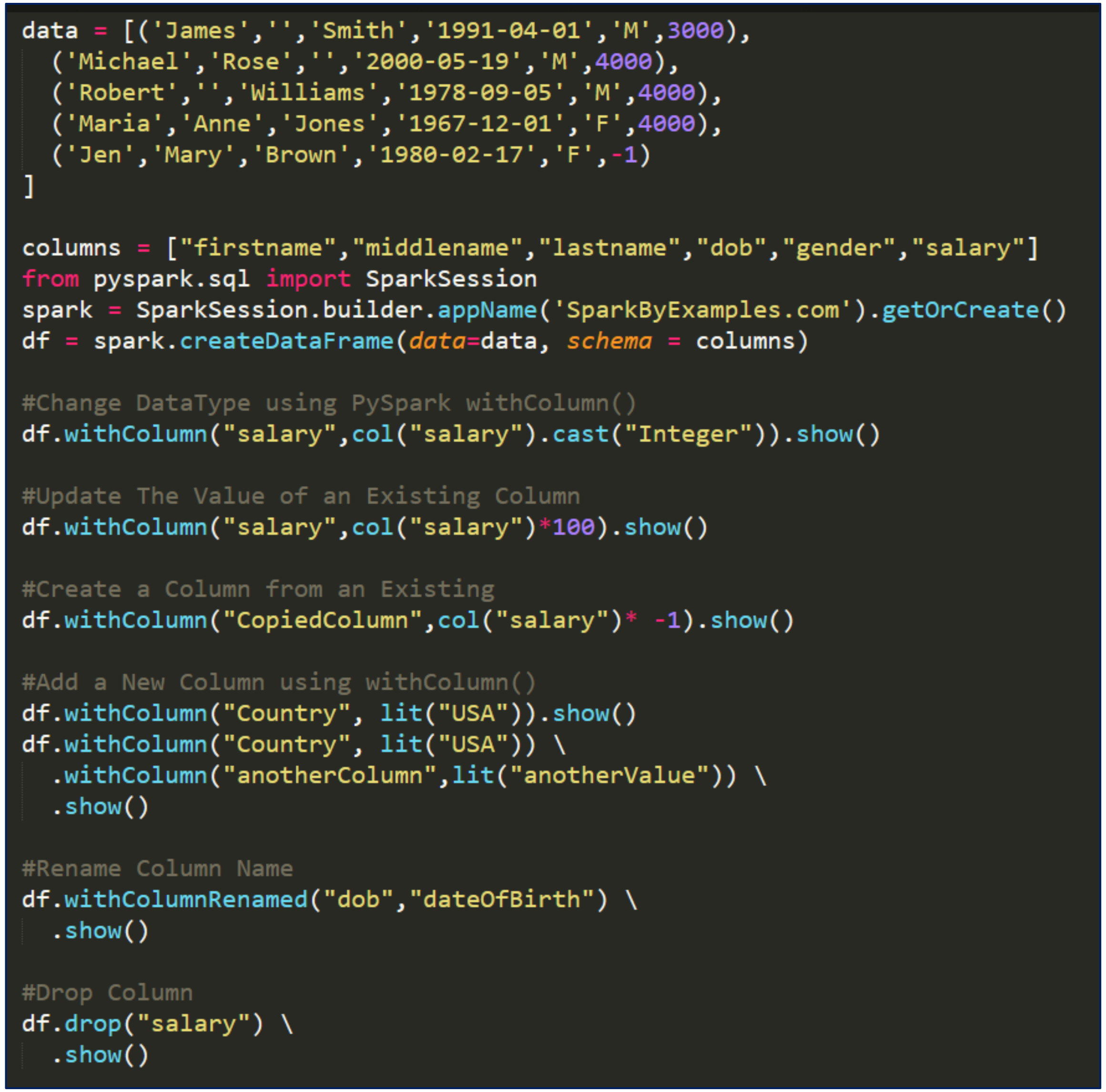


```
data = [("James","Smith","USA","CA"),
        ("Michael","Rose","USA","NY"),
        ("Robert","Williams","USA","CA"),
        ("Maria","Jones","USA","FL")
       ]
columns = ["firstname","lastname","country","state"]
df = spark.createDataFrame(data = data, schema = columns)
df.show() #display(df)

df.select("firstname","lastname").show()
df.select(df.firstname,df.lastname).show()
df.select(df["firstname"],df["lastname"]).show()

#By using col() function
from pyspark.sql.functions import col
df.select(col("firstname"),col("lastname")).show()

#Select columns by regular expression
df.select(df.colRegex("`^.*name*`")).show()

# Select All columns from List
df.select(*columns).show()

# Select All columns
df.select([col for col in df.columns]).show()
df.select("*").show()

#Selects first 3 columns and top 3 rows
df.select(df.columns[:3]).show(3)

#Selects columns 2 to 4  and top 3 rows
df.select(df.columns[2:4]).show(3)
```

Fuction :
select() to select column in specific Dataframe

*Similar to select in SQL

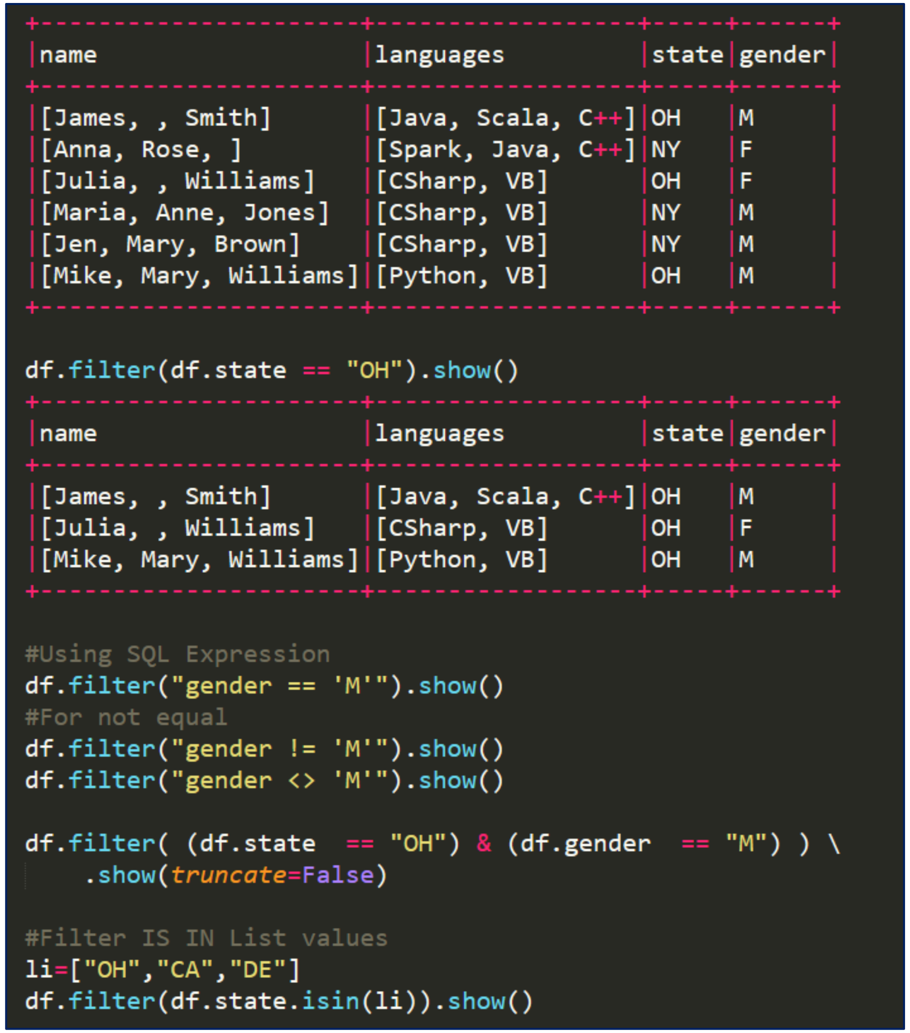
8. Pyspark Basic Command



```
data = [('James','','Smith','1991-04-01','M',3000),
  ('Michael','Rose','','2000-05-19','M',4000),
  ('Robert','','Williams','1978-09-05','M',4000),
  ('Maria','Anne','Jones','1967-12-01','F',4000),
  ('Jen','Mary','Brown','1980-02-17','F',-1)
]

columns = ["firstname","middlename","lastname","dob","gender","salary"]
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
df = spark.createDataFrame(data=data, schema = columns)

#Change DataType using PySpark withColumn()
df.withColumn("salary",col("salary").cast("Integer")).show()

#Update The Value of an Existing Column
df.withColumn("salary",col("salary")*100).show()

#Create a Column from an Existing
df.withColumn("CopiedColumn",col("salary")* -1).show()

#Add a New Column using withColumn()
df.withColumn("Country", lit("USA")).show()
df.withColumn("Country", lit("USA")) \
  .withColumn("anotherColumn",lit("anotherValue")) \
  .show()

#Rename Column Name
df.withColumnRenamed("dob","dateOfBirth") \
  .show()

#Drop Column
df.drop("salary") \
  .show()
```

Fuction :
withColumn() to manipulate data in column

8. Pyspark Basic Command

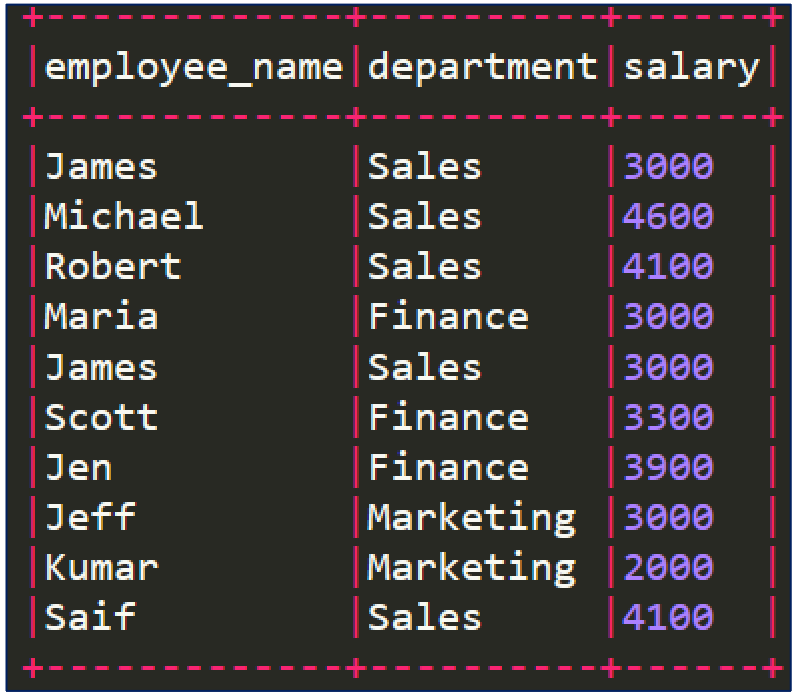


```
+-----------------------+------------------+-----+------+
|name                   |languages         |state|gender|
+-----------------------+------------------+-----+------+
|[James, , Smith]       |[Java, Scala, C++]|OH   |M     |
|[Anna, Rose, ]         |[Spark, Java, C++]|NY   |F     |
|[Julia, , Williams]    |[CSharp, VB]      |OH   |F     |
|[Maria, Anne, Jones]   |[CSharp, VB]      |NY   |M     |
|[Jen, Mary, Brown]     |[CSharp, VB]      |NY   |M     |
|[Mike, Mary, Williams] |[Python, VB]      |OH   |M     |
+-----------------------+------------------+-----+------+

df.filter(df.state == "OH").show()
+-----------------------+------------------+-----+------+
|name                   |languages         |state|gender|
+-----------------------+------------------+-----+------+
|[James, , Smith]       |[Java, Scala, C++]|OH   |M     |
|[Julia, , Williams]    |[CSharp, VB]      |OH   |F     |
|[Mike, Mary, Williams] |[Python, VB]      |OH   |M     |
+-----------------------+------------------+-----+------+


#Using SQL Expression
df.filter("gender == 'M'").show()
#For not equal
df.filter("gender != 'M'").show()
df.filter("gender <> 'M'").show()

df.filter( (df.state  == "OH") & (df.gender  == "M") ) \
    .show(truncate=False)

#Filter IS IN List values
li=["OH","CA","DE"]
df.filter(df.state.isin(li)).show()
```

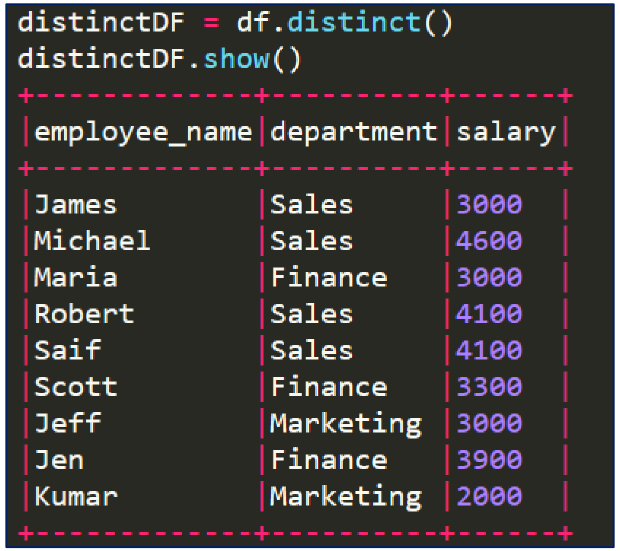
Fuction :
filter() to filtering data with Condition in Dataframe

*Similar to where in SQL

8. Pyspark Basic Command



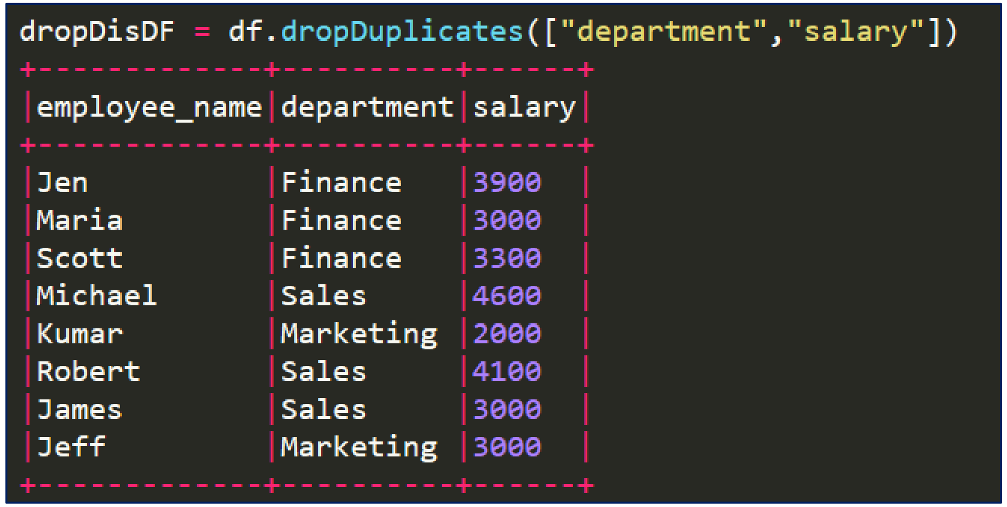
```
+-------------+----------+------+
|employee_name|department|salary|
+-------------+----------+------+
|James        |Sales     |3000  |
|Michael      |Sales     |4600  |
|Robert       |Sales     |4100  |
|Maria        |Finance   |3000  |
|James        |Sales     |3000  |
|Scott        |Finance   |3300  |
|Jen          |Finance   |3900  |
|Jeff         |Marketing |3000  |
|Kumar        |Marketing |2000  |
|Saif         |Sales     |4100  |
+-------------+----------+------+
```



```
distinctDF = df.distinct()
distinctDF.show()
+-------------+----------+------+
|employee_name|department|salary|
+-------------+----------+------+
|James        |Sales     |3000  |
|Michael      |Sales     |4600  |
|Maria        |Finance   |3000  |
|Robert       |Sales     |4100  |
|Saif         |Sales     |4100  |
|Scott        |Finance   |3300  |
|Jeff         |Marketing |3000  |
|Jen          |Finance   |3900  |
|Kumar        |Marketing |2000  |
+-------------+----------+------+
```

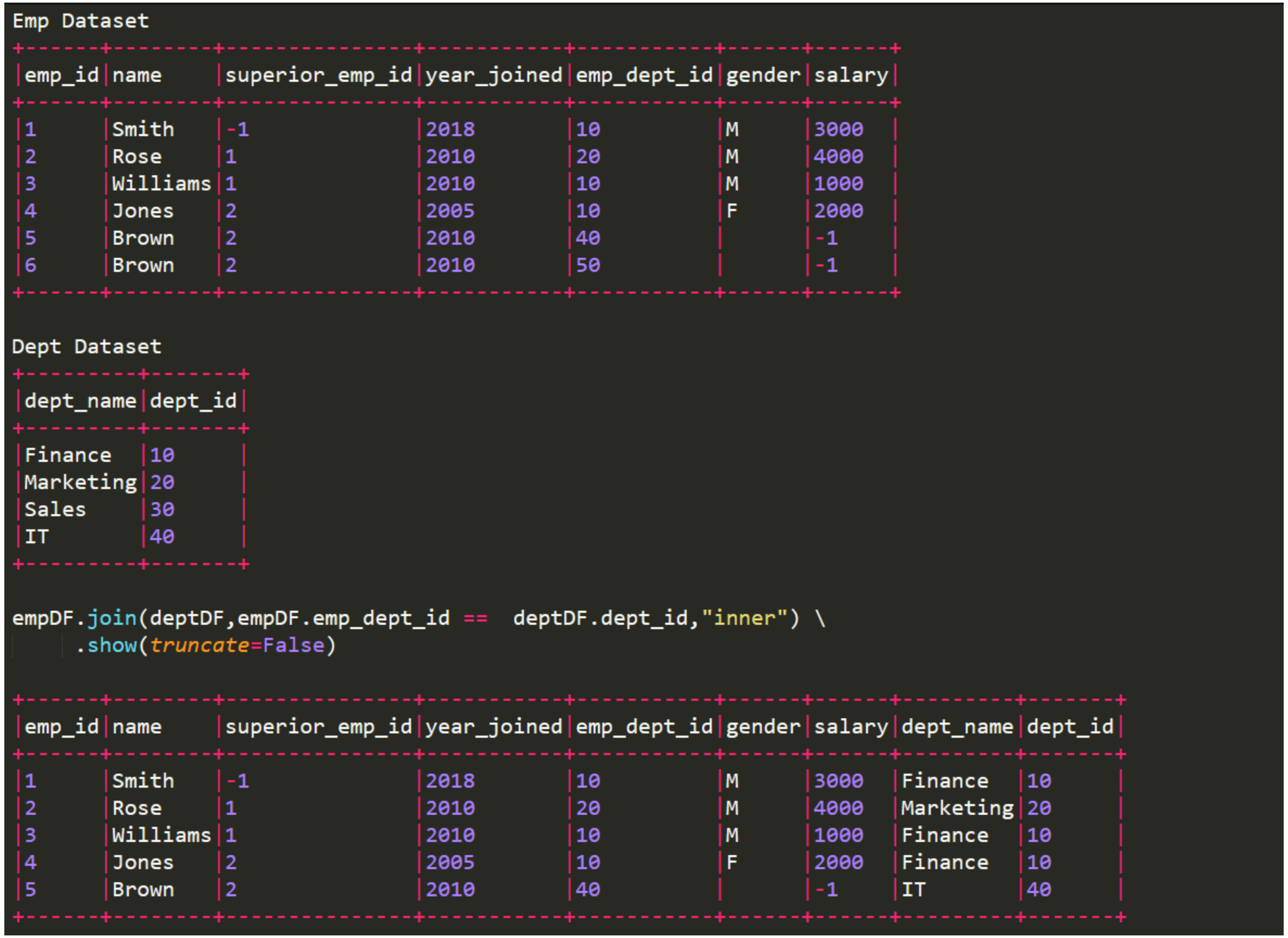
Fuction :
distinct() / dropDuplicates()
to remove duplicate row.

*Similar to ‘select distinct’ in SQL



```
dropDisDF = df.dropDuplicates(["department","salary"])
+-------------+----------+------+
|employee_name|department|salary|
+-------------+----------+------+
|Jen          |Finance   |3900  |
|Maria        |Finance   |3000  |
|Scott        |Finance   |3300  |
|Michael      |Sales     |4600  |
|Kumar        |Marketing |2000  |
|Robert       |Sales     |4100  |
|James        |Sales     |3000  |
|Jeff         |Marketing |3000  |
+-------------+----------+------+
```

8. Pyspark Basic Command



Emp Dataset

```
+------+--------+---------------+-----------+-----------+------+------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|
+------+--------+---------------+-----------+-----------+------+------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |
|2     |Rose    |1              |2010       |20         |M     |4000  |
|3     |Williams|1              |2010       |10         |M     |1000  |
|4     |Jones   |2              |2005       |10         |F     |2000  |
|5     |Brown   |2              |2010       |40         |      |-1    |
|6     |Brown   |2              |2010       |50         |      |-1    |
+------+--------+---------------+-----------+-----------+------+------+
```

Dept Dataset

```
+---------+-------+
|dept_name|dept_id|
+---------+-------+
|Finance  |10     |
|Marketing|20     |
|Sales    |30     |
|IT       |40     |
+---------+-------+
```

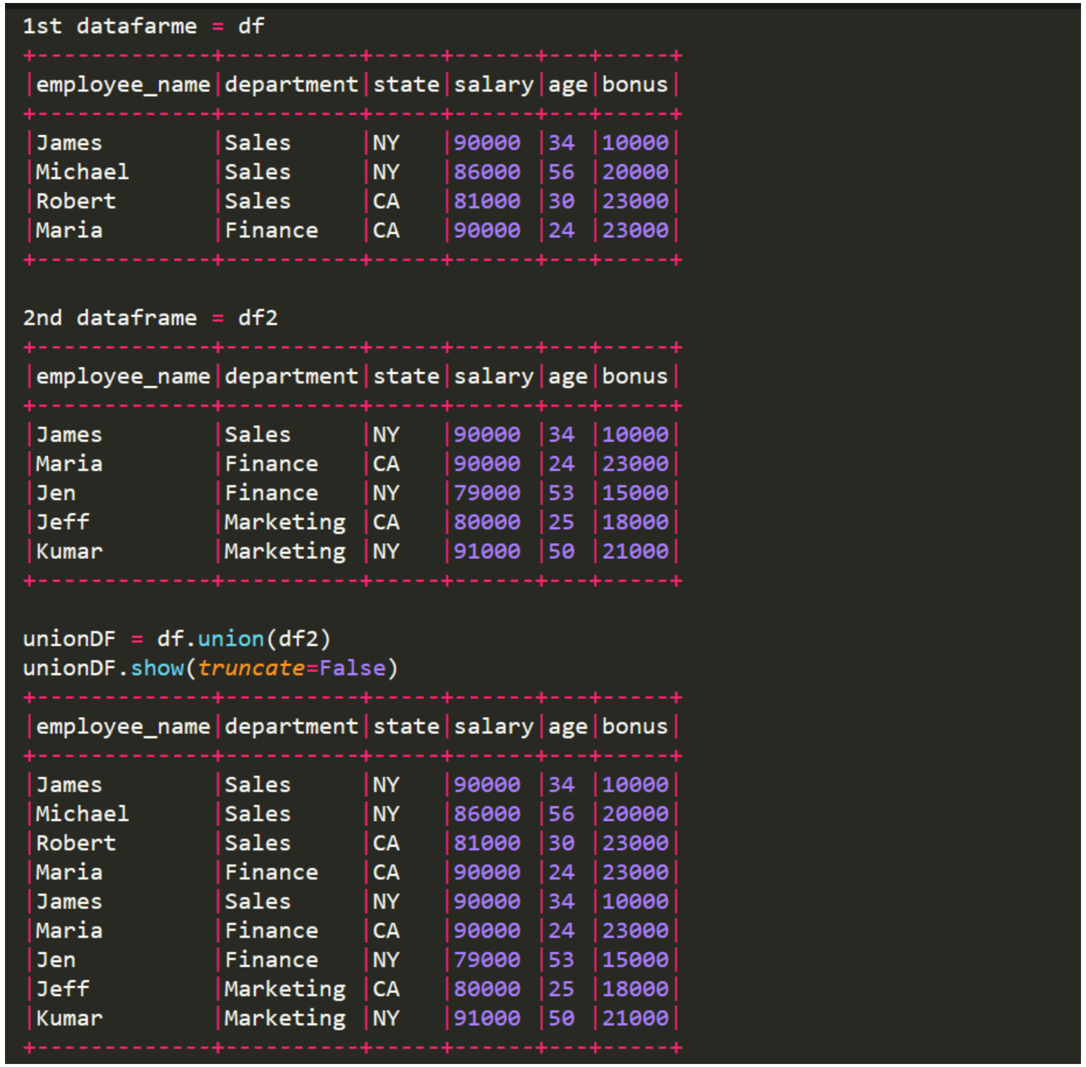
```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") \
     .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

Fuction :
join() data between 2 dataframe or self join

*Similar to join in SQL

8. Pyspark Basic Command

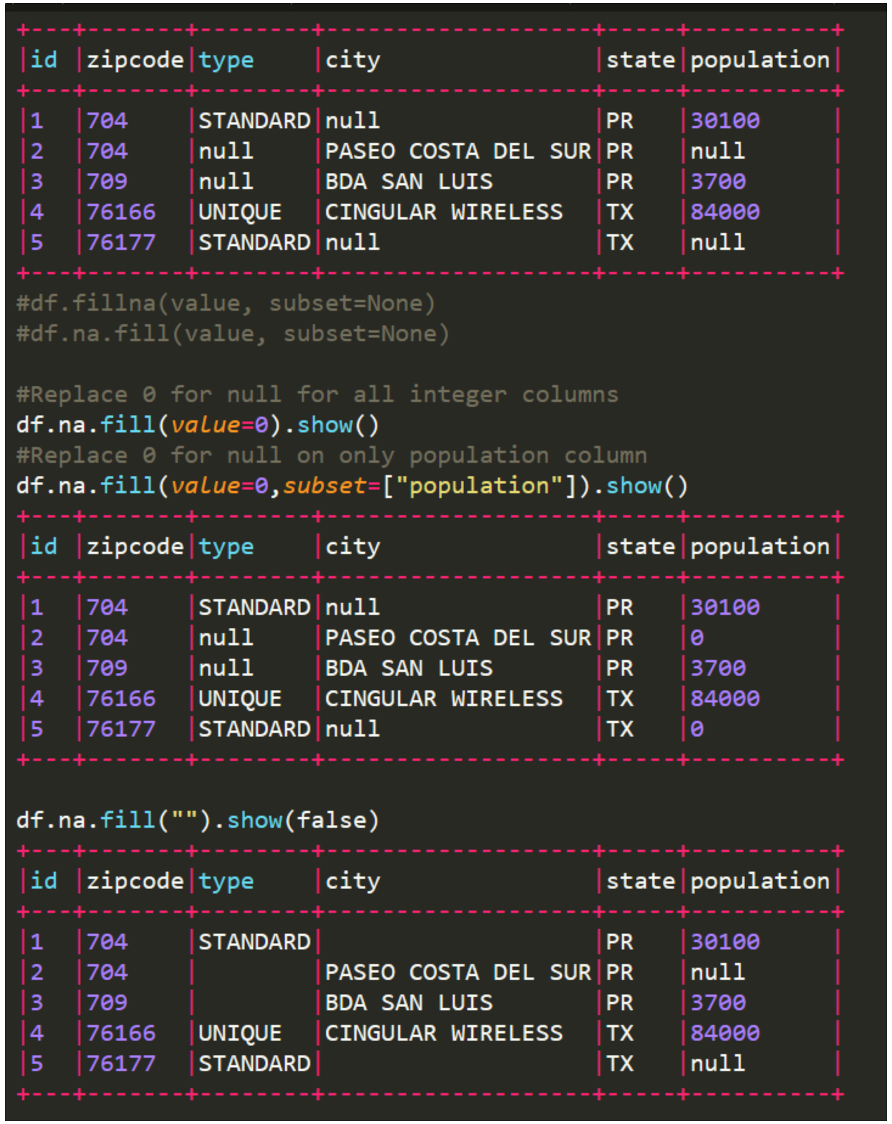


```
1st datafarme = df
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Michael      |Sales     |NY   |86000 |56 |20000|
|Robert       |Sales     |CA   |81000 |30 |23000|
|Maria        |Finance   |CA   |90000 |24 |23000|
+-------------+----------+-----+------+---+-----+

2nd dataframe = df2
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|Jen          |Finance   |NY   |79000 |53 |15000|
|Jeff         |Marketing |CA   |80000 |25 |18000|
|Kumar        |Marketing |NY   |91000 |50 |21000|
+-------------+----------+-----+------+---+-----+

unionDF = df.union(df2)
unionDF.show(truncate=False)
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Michael      |Sales     |NY   |86000 |56 |20000|
|Robert       |Sales     |CA   |81000 |30 |23000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|James        |Sales     |NY   |90000 |34 |10000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|Jen          |Finance   |NY   |79000 |53 |15000|
|Jeff         |Marketing |CA   |80000 |25 |18000|
|Kumar        |Marketing |NY   |91000 |50 |21000|
+-------------+----------+-----+------+---+-----+
```

Fuction :
union() use to merge data between 2 dataframes.

*Similar to union in SQL

8. Pyspark Basic Command

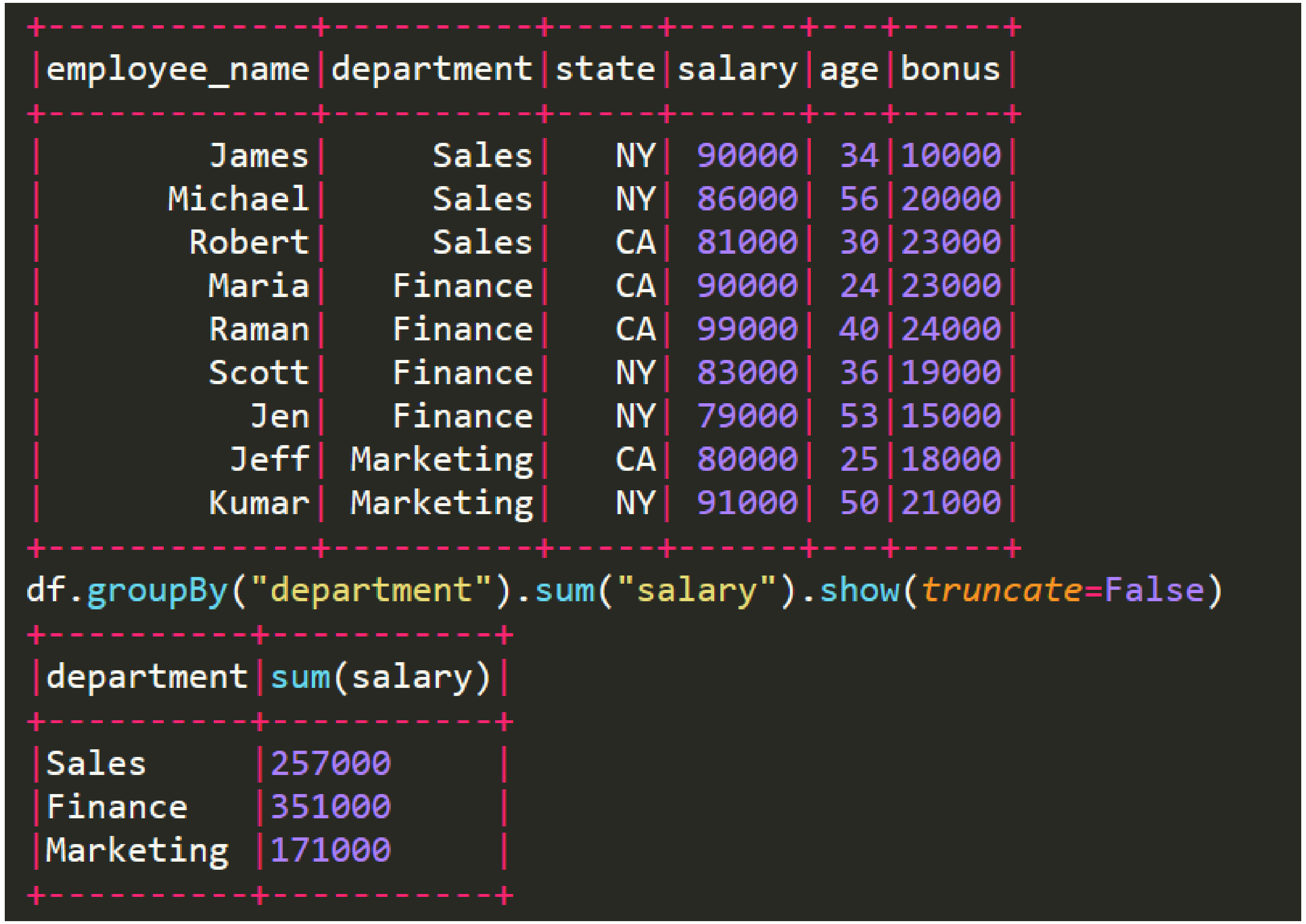


```
+---+-------+--------+-------------------+-----+----------+
|id |zipcode|type    |city               |state|population|
+---+-------+--------+-------------------+-----+----------+
|1  |704    |STANDARD|null               |PR   |30100     |
|2  |704    |null    |PASEO COSTA DEL SUR|PR   |null      |
|3  |709    |null    |BDA SAN LUIS       |PR   |3700      |
|4  |76166  |UNIQUE  |CINGULAR WIRELESS  |TX   |84000     |
|5  |76177  |STANDARD|null               |TX   |null      |
+---+-------+--------+-------------------+-----+----------+
#df.fillna(value, subset=None)
#df.na.fill(value, subset=None)

#Replace 0 for null for all integer columns
df.na.fill(value=0).show()
#Replace 0 for null on only population column
df.na.fill(value=0,subset=["population"]).show()
+---+-------+--------+-------------------+-----+----------+
|id |zipcode|type    |city               |state|population|
+---+-------+--------+-------------------+-----+----------+
|1  |704    |STANDARD|null               |PR   |30100     |
|2  |704    |null    |PASEO COSTA DEL SUR|PR   |0         |
|3  |709    |null    |BDA SAN LUIS       |PR   |3700      |
|4  |76166  |UNIQUE  |CINGULAR WIRELESS  |TX   |84000     |
|5  |76177  |STANDARD|null               |TX   |0         |
+---+-------+--------+-------------------+-----+----------+

df.na.fill("").show(false)
+---+-------+--------+-------------------+-----+----------+
|id |zipcode|type    |city               |state|population|
+---+-------+--------+-------------------+-----+----------+
|1  |704    |STANDARD|                   |PR   |30100     |
|2  |704    |        |PASEO COSTA DEL SUR|PR   |null      |
|3  |709    |        |BDA SAN LUIS       |PR   |3700      |
|4  |76166  |UNIQUE  |CINGULAR WIRELESS  |TX   |84000     |
|5  |76177  |STANDARD|                   |TX   |null      |
+---+-------+--------+-------------------+-----+----------+
```

Fuction :
na.fill() to replace null with specific values

*Similar to coalesce / isnull in SQL

8. Pyspark Basic Command



```
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|        James|     Sales|   NY| 90000| 34|10000|
|      Michael|     Sales|   NY| 86000| 56|20000|
|       Robert|     Sales|   CA| 81000| 30|23000|
|        Maria|   Finance|   CA| 90000| 24|23000|
|        Raman|   Finance|   CA| 99000| 40|24000|
|        Scott|   Finance|   NY| 83000| 36|19000|
|          Jen|   Finance|   NY| 79000| 53|15000|
|         Jeff| Marketing|   CA| 80000| 25|18000|
|        Kumar| Marketing|   NY| 91000| 50|21000|
+-------------+----------+-----+------+---+-----+
df.groupBy("department").sum("salary").show(truncate=False)
+----------+-----------+
|department|sum(salary)|
+----------+-----------+
|Sales     |257000     |
|Finance   |351000     |
|Marketing |171000     |
+----------+-----------+
```

Fuction :
groupBy() to perform count, sum, avg, min, max functions on the grouped data.
*Similar to group by in SQL