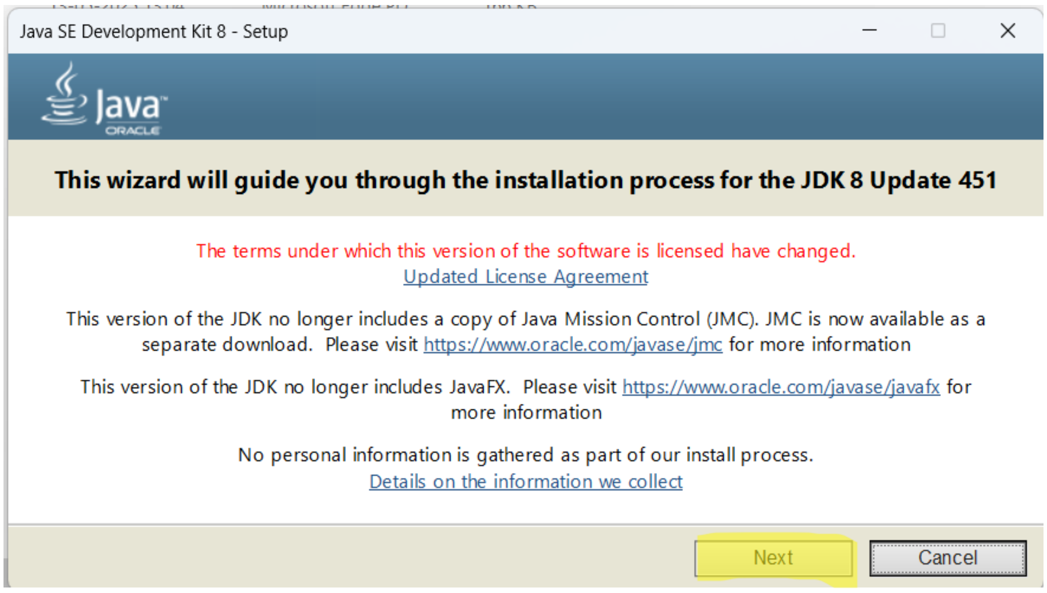
Practical :- 07

Aim: Install, configure and run Hadoop and HDFS and explore HDFS.

Description:

This lab guides through the process of setting up a local Hadoop environment on a Windows machine. It includes installing Java, downloading Hadoop, setting up system environment variables, configuring core Hadoop XML files, and verifying the setup.

Step 1: Install java (jdk)



Java SE Development Kit 8 - Setup

This wizard will guide you through the installation process for the JDK 8 Update 451

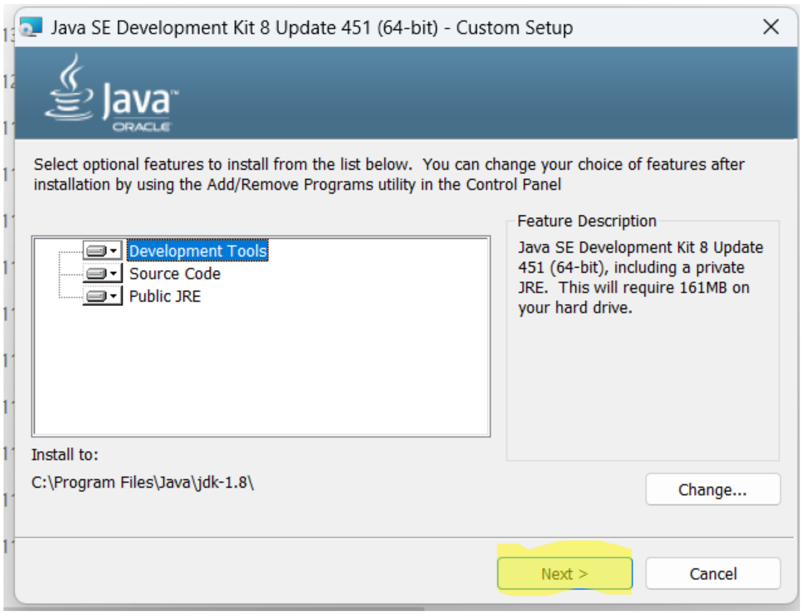
The terms under which this version of the software is licensed have changed.
Updated License Agreement

This version of the JDK no longer includes a copy of Java Mission Control (JMC). JMC is now available as a separate download. Please visit https://www.oracle.com/javase/jmc for more information

This version of the JDK no longer includes JavaFX. Please visit https://www.oracle.com/javase/javafx for more information

No personal information is gathered as part of our install process.
Details on the information we collect

Click next (Create a java folder in drive)



Java SE Development Kit 8 Update 451 (64-bit) - Custom Setup

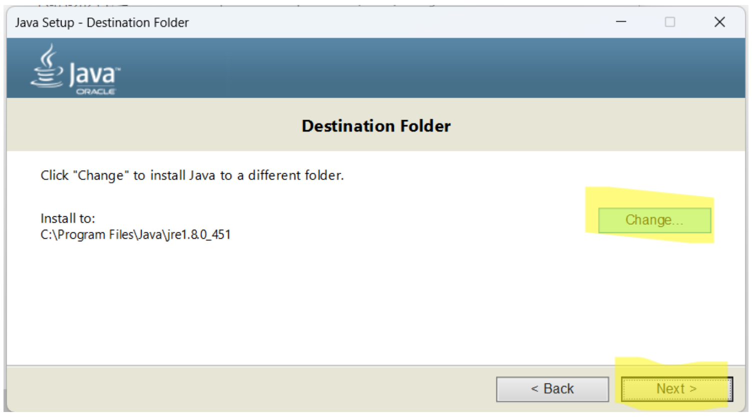
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

Development Tools
Source Code
Public JRE

Feature Description
Java SE Development Kit 8 Update 451 (64-bit), including a private JRE. This will require 161MB on your hard drive.

Install to:
C:\Program Files\Java\jdk-1.8\

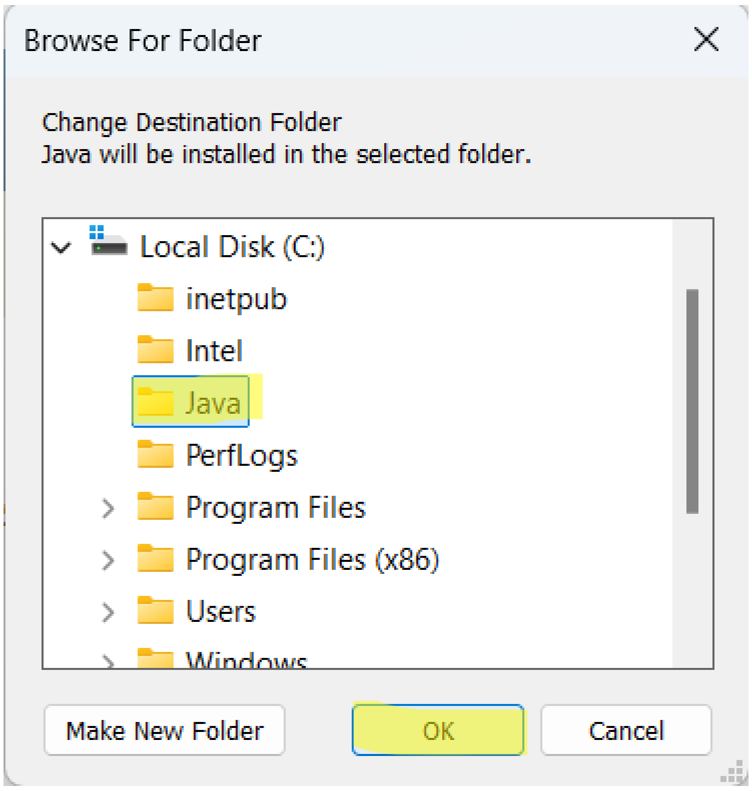
Click next



Click "Change" to install Java to a different folder.

Install to:
C:\Program Files\Java\jre1.8.0_451

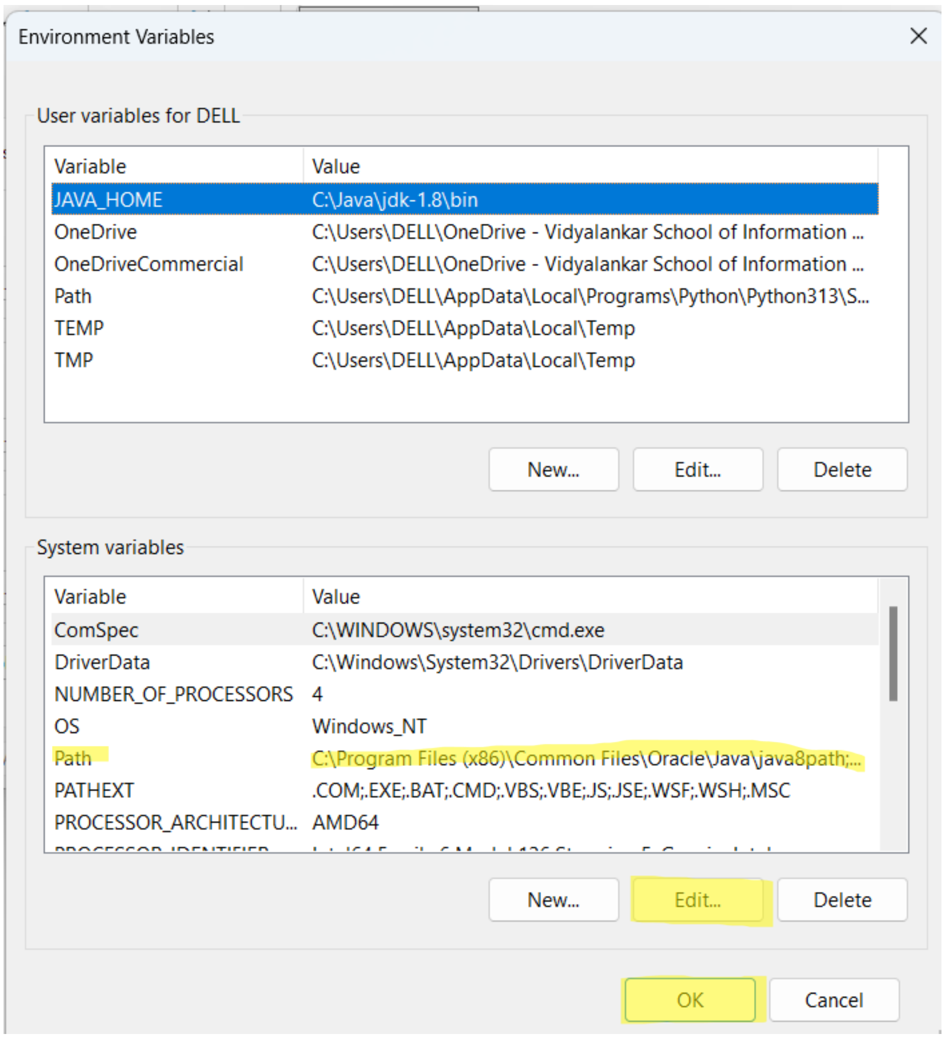
change the path to Java in C drive



Change Destination Folder
Java will be installed in the selected folder.

Click on ok and then next

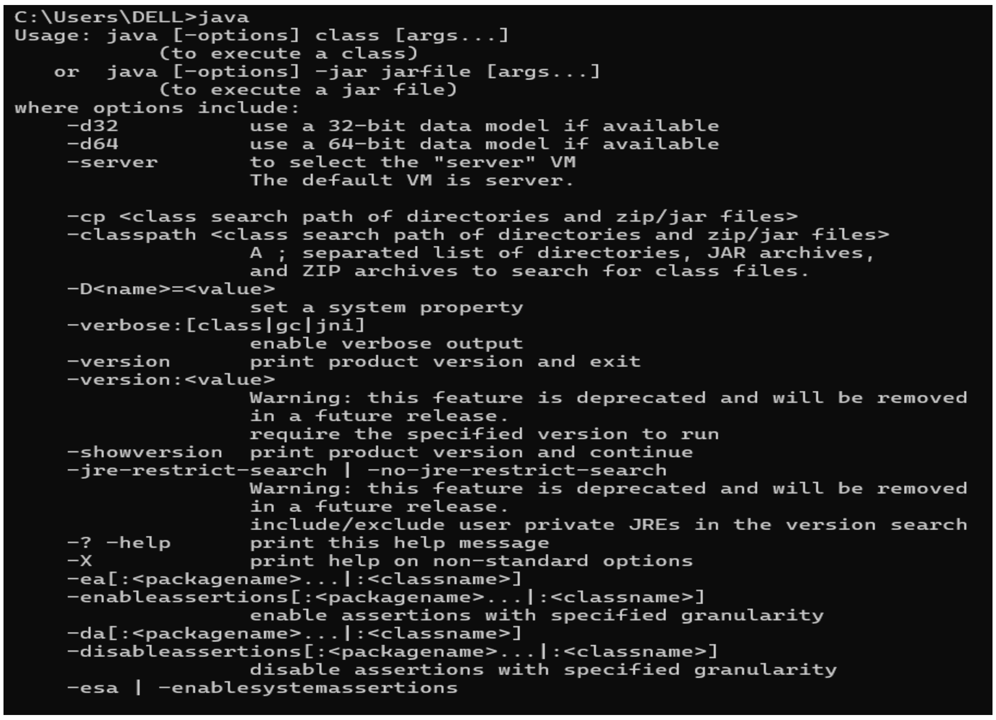
This PC -> Program files cut the jdk folder and paste that folder in the java folder created by you -> Edit system variables -> click on new -> Variable name : JAVA_HOME -> Path: C:\Java\jdk-1.8\bin-> In system variables below click on path then click on edit and paste the path (C:\Java\jdk-1.8\bin)-> ok



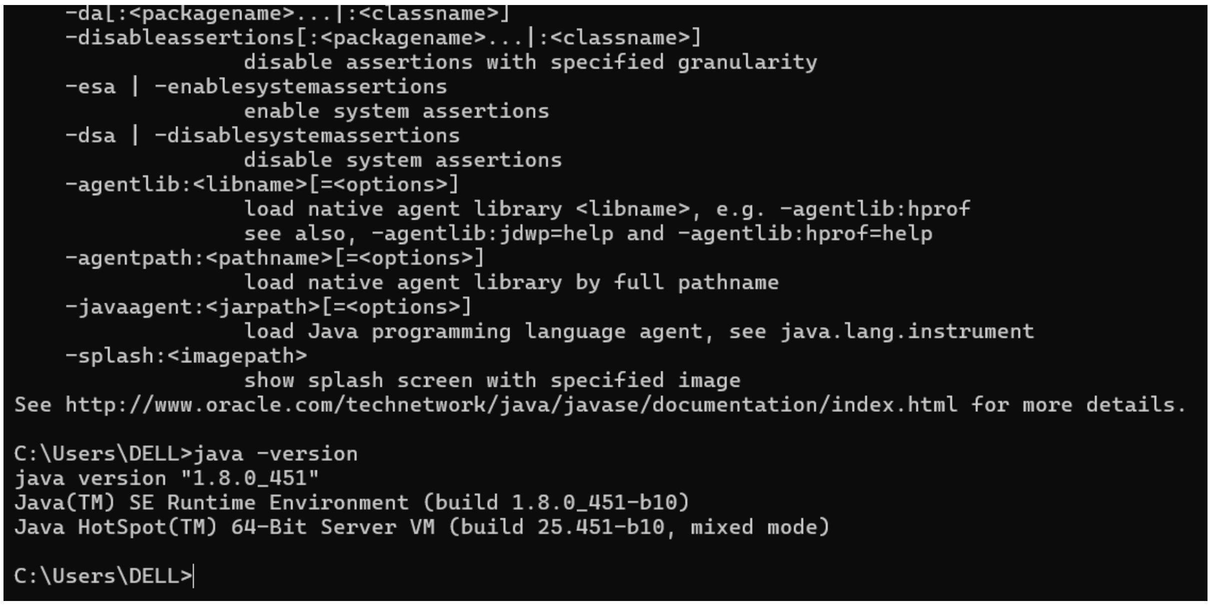
Variable	Value
JAVA_HOME	C:\Java\jdk-1.8\bin
OneDrive	C:\Users\DELL\OneDrive - Vidyalankar School of Information ...
OneDriveCommercial	C:\Users\DELL\OneDrive - Vidyalankar School of Information ...
Path	C:\Users\DELL\AppData\Local\Programs\Python\Python313\S...
TEMP	C:\Users\DELL\AppData\Local\Temp
TMP	C:\Users\DELL\AppData\Local\Temp

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\java8path;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64

To verify use command prompt



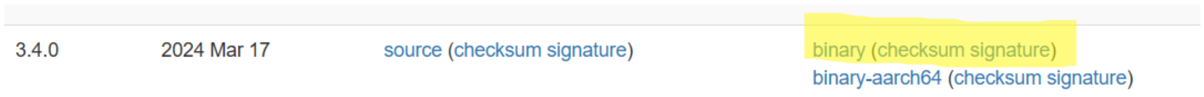
```
C:\Users\DELL>java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  include/exclude user private JREs in the version search
    -? -help      print this help message
    -X            print help on non-standard options
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions with specified granularity
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions with specified granularity
    -esa | -enablesystemassertions
```



```
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions with specified granularity
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    -agentlib:<libname>[=<options>]
                  load native agent library <libname>, e.g. -agentlib:hprof
                  see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                  load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                  load Java programming language agent, see java.lang.instrument
    -splash:<imagepath>
                  show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for more details.

C:\Users\DELL>java -version
java version "1.8.0_451"
Java(TM) SE Runtime Environment (build 1.8.0_451-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.451-b10, mixed mode)

C:\Users\DELL>
```

Step 2: Install Hadoop

Browser -> search “apache Hadoop” -> click on the first link (https://hadoop.apache.org) -> click on download

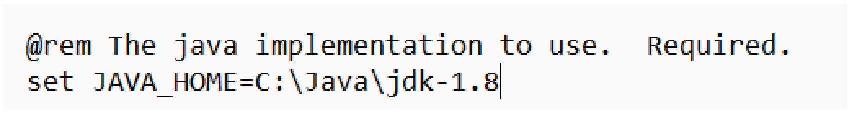


3.4.0	2024 Mar 17	source (checksum signature)	binary (checksum signature) binary-aarch64 (checksum signature)

Click on https://dlcdn.apache.org/hadoop/common/hadoop-3.4.0/hadoop-3.4.0.tar.gz

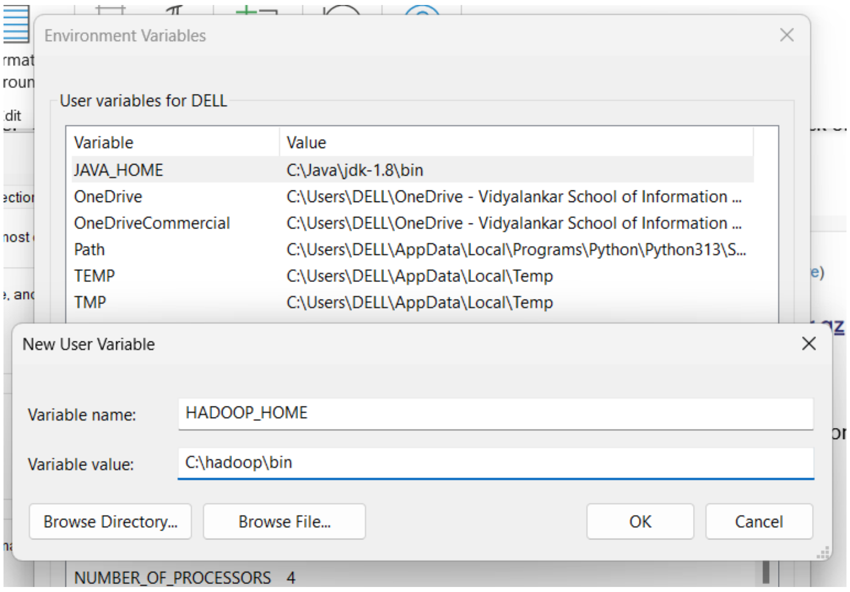
After download is done extract the files in C drive and rename the folder as hadoop

Open the hadoop folder -> click on etc -> -> click on hadoop -> right click on hadoop-env -> click on edit -> change the java jdk path



```
@rem The java implementation to use.  Required.
set JAVA_HOME=C:\Java\jdk-1.8
```

Click on edit system variables ->



Environment Variables

User variables for DELL

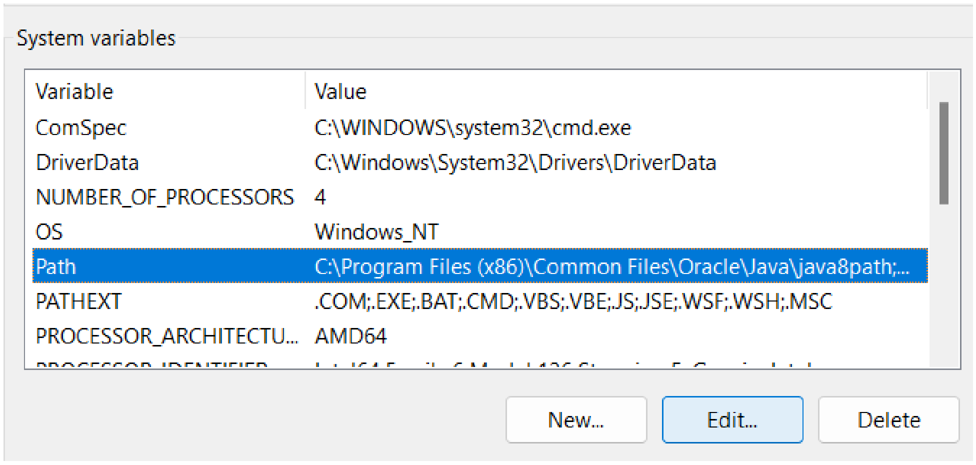
Variable	Value
JAVA_HOME	C:\Java\jdk-1.8\bin
OneDrive	C:\Users\DELL\OneDrive - Vidyalankar School of Information ...
OneDriveCommercial	C:\Users\DELL\OneDrive - Vidyalankar School of Information ...
Path	C:\Users\DELL\AppData\Local\Programs\Python\Python313\S...
TEMP	C:\Users\DELL\AppData\Local\Temp
TMP	C:\Users\DELL\AppData\Local\Temp

New User Variable

Variable name: HADOOP_HOME

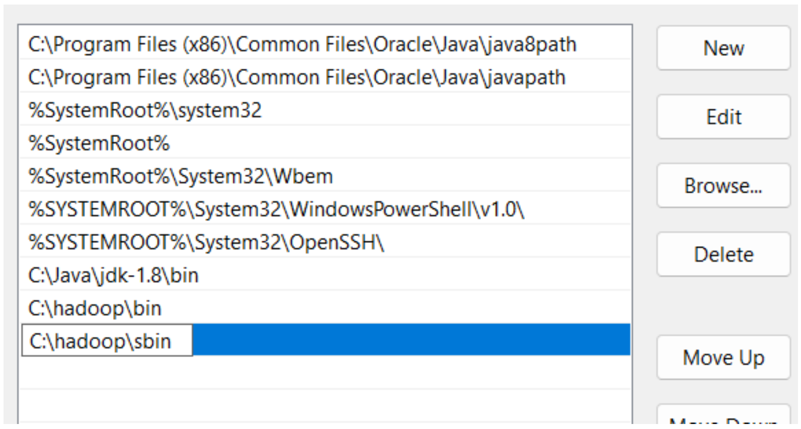
Variable value: C:\hadoop\bin

NUMBER_OF_PROCESSORS 4



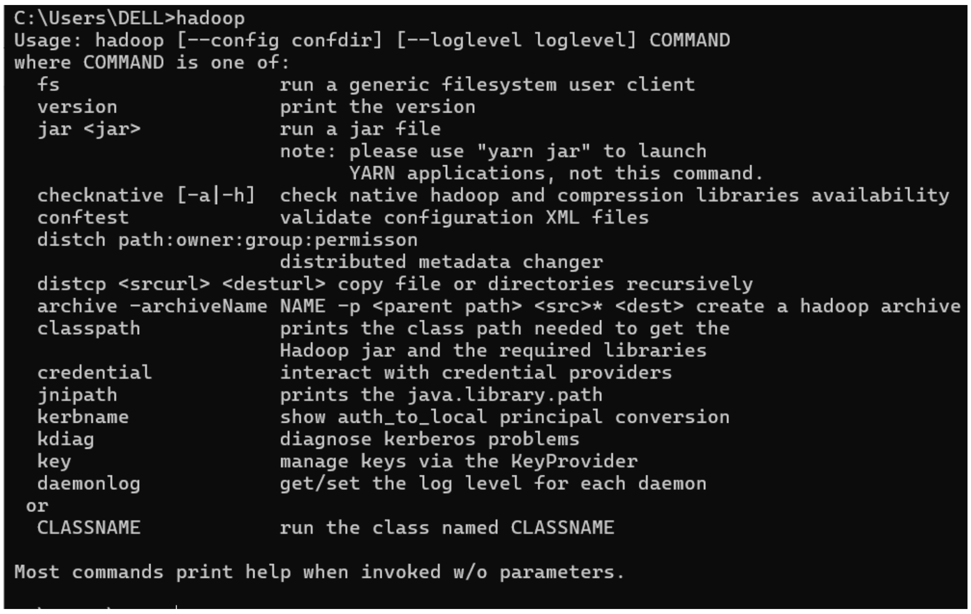
System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\java8path;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64

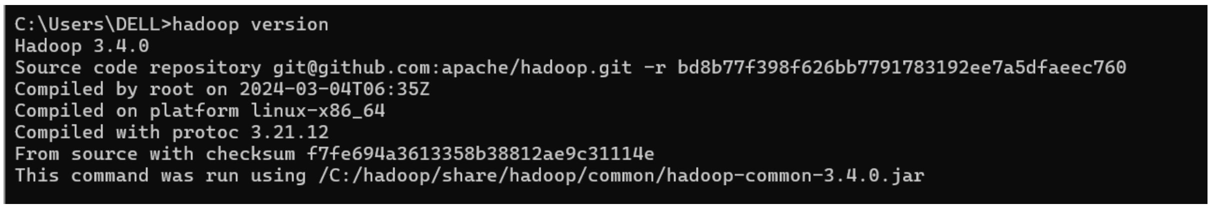


```
C:\Program Files (x86)\Common Files\Oracle\Java\java8path
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Java\jdk-1.8\bin
C:\hadoop\bin
C:\hadoop\sbin
```

To verify open command prompt -> type Hadoop



```
C:\Users\DELL>hadoop
Usage: hadoop [--config confdir] [--loglevel loglevel] COMMAND
where COMMAND is one of:
  fs                   run a generic filesystem user client
  version              print the version
  jar <jar>            run a jar file
                       note: please use "yarn jar" to launch
                             YARN applications, not this command.
  checknative [-a|-h]  check native hadoop and compression libraries availability
  conftest             validate configuration XML files
  distch path:owner:group:permisson
                       distributed metadata changer
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
  classpath            prints the class path needed to get the
                       Hadoop jar and the required libraries
  credential           interact with credential providers
  jnipath              prints the java.library.path
  kerbname             show auth_to_local principal conversion
  kdiag                diagnose kerberos problems
  key                  manage keys via the KeyProvider
  daemonlog            get/set the log level for each daemon
 or
  CLASSNAME            run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
```



```
C:\Users\DELL>hadoop version
Hadoop 3.4.0
Source code repository git@github.com:apache/hadoop.git -r bd8b77f398f626bb7791783192ee7a5dfaeec760
Compiled by root on 2024-03-04T06:35Z
Compiled on platform linux-x86_64
Compiled with protoc 3.21.12
From source with checksum f7fe694a3613358b38812ae9c31114e
This command was run using /C:/hadoop/share/hadoop/common/hadoop-common-3.4.0.jar
```

Step 3: Configure Hadoop to run in our system

Hadoop -> etc -> Hadoop -> Right click on the core-site.xml

In core-site.xml add the property in configuration tag

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>

Hadoop -> etc -> Hadoop -> Right click on the httpfs-site.xml

Go to Hadoop folder -> create a new folder “data” -> Inside the data folder create two more folders -> “namenode” and “datanode”

In httpfs-site.xml add 3 property tags in configuration tag

<configuration>

<property>

<name>dfs.replication</name>

```
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>C:\hadoop\data\namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>C:\hadoop\data\datanode</value>
</property>
</configuration>
```

Hadoop -> etc -> Hadoop -> Right click on the mapred-site.xml

In mapred-site.xml add a property tag in configuration tag

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

Hadoop -> etc -> Hadoop -> Right click on the yarn-site.xml

In yarn-site.xml add 2 property tags in configuration tag

```
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
```

```
<property>
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>
```

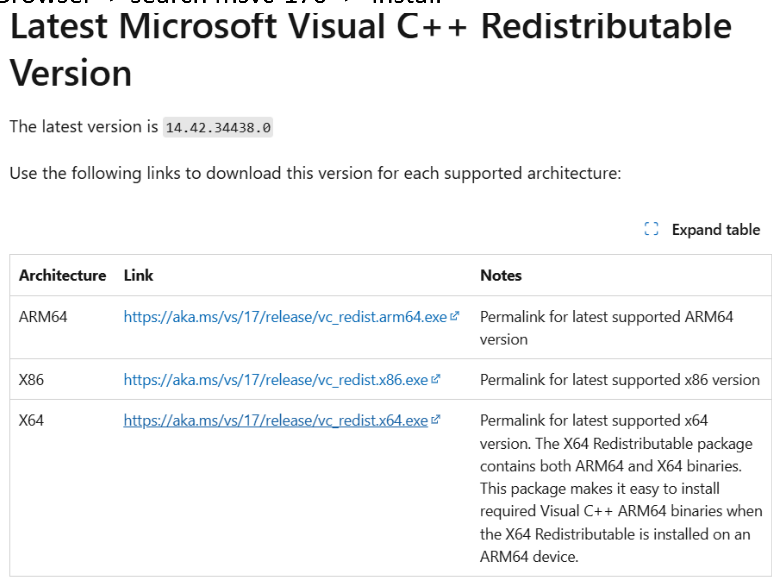
Now we need to fix the bin folder to run Hadoop in windows

Open Hadoop folder -> delete the bin folder -> download the bin folder with this link "hadoop3_xFixedbin.rar - Google Drive"

Browser install msvcr120 dll download -> download it in this PC-> C drive -> Windows -> system32 folder -> run the winutils file in Hadoop's bin folder -> there should be no error

Step 4: install c++

Browser -> search msvc-170 -> install



Latest Microsoft Visual C++ Redistributable Version

The latest version is `14.42.34438.0`

Use the following links to download this version for each supported architecture:

Architecture	Link	Notes
ARM64	https://aka.ms/vs/17/release/vc_redist.arm64.exe	Permalink for latest supported ARM64 version
X86	https://aka.ms/vs/17/release/vc_redist.x86.exe	Permalink for latest supported x86 version
X64	https://aka.ms/vs/17/release/vc_redist.x64.exe	Permalink for latest supported x64 version. The X64 Redistributable package contains both ARM64 and X64 binaries. This package makes it easy to install required Visual C++ ARM64 binaries when the X64 Redistributable is installed on an ARM64 device.

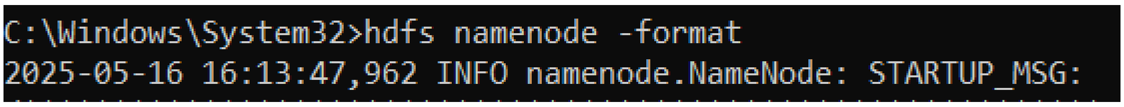
run command prompt as administrator ->

Commands

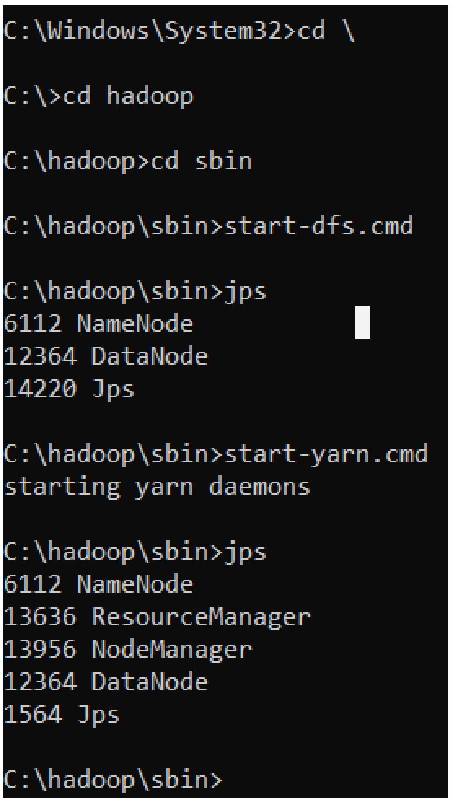
```
hdfs namenode -format
cd \
cd hadoop
cd sbin
start-dfs.cmd
jps
```

start-yarn.cmd

jps



```
C:\Windows\System32>hdfs namenode -format
2025-05-16 16:13:47,962 INFO namenode.NameNode: STARTUP_MSG:
```



```
C:\Windows\System32>cd \

C:\>cd hadoop

C:\hadoop>cd sbin

C:\hadoop\sbin>start-dfs.cmd

C:\hadoop\sbin>jps
6112 NameNode
12364 DataNode
14220 Jps

C:\hadoop\sbin>start-yarn.cmd
starting yarn daemons

C:\hadoop\sbin>jps
6112 NameNode
13636 ResourceManager
13956 NodeManager
12364 DataNode
1564 Jps

C:\hadoop\sbin>
```

Learnings / Conclusion: After completing this lab, the user will be able to successfully install and configure Apache Hadoop on a Windows environment. They will also learn to set up the environment variables, edit Hadoop configuration files, and start Hadoop services using command line utilities.

Practical :- 08

a) Aim: Implement word count / frequency program.

Description:
A Word Count / Frequency Program using a simple MapReduce approach is a classic example that demonstrates how the MapReduce programming model works by processing large datasets (like text) in a distributed and parallel manner. In this context, the goal is to count how often each word appears in a given set of text data.

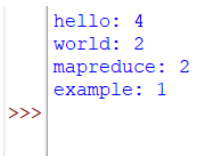
MapReduce is a programming model used to process and generate large datasets in parallel and distributed environments. It consists of two main phases:

1. **Map Phase**:
   - The input data is divided into chunks, and each chunk is processed by a "mapper."
   - The mapper takes the input data and produces a set of key-value pairs. For example, if you're counting words, the mapper would take each word in a text line and emit (word, 1)—i.e., each word is paired with the number 1 to indicate one occurrence.
2. **Reduce Phase**:
   - The output of the mappers (the key-value pairs) is grouped by key. For example, all occurrences of the same word are grouped together.
   - The reducer then processes these grouped pairs. In the case of word count, the reducer sums the values (the 1s) for each word to get the total count for that word.

Code:
```
from collections import defaultdict

# Map function: takes a line of text, splits it into words, and outputs a list of key-value pairs
def map_function(text):
    words = text.split()
    for word in words:
        yield (word.lower(), 1)  # Emit word and count 1

# Reduce function: takes the key and a list of values (counts), sums the counts and returns the total
def reduce_function(word, counts):
    return (word, sum(counts))

# Driver function that simulates a MapReduce process
def word_count_map_reduce(text_data):
    # Map step
    intermediate_data = defaultdict(list)
    for line in text_data:
        for word, count in map_function(line):
            intermediate_data[word].append(count)

    # Reduce step
    result = {}
    for word, counts in intermediate_data.items():
        result[word] = reduce_function(word, counts)

    return result
```

```
# Example input (text_data)
text_data = [
    "hello world",
    "hello mapreduce",
    "mapreduce example",
    "hello hello world"
]

# Run the MapReduce simulation
word_counts = word_count_map_reduce(text_data)

# Output result
for word, count in word_counts.items():
    print(f"{word}: {count[1]}")
```

Output:



```
hello: 4
world: 2
mapreduce: 2
example: 1
>>>
```

Learnings:

Working

- **Map Phase**: The mapper reads the data (lines of text) and outputs key-value pairs (word, 1).
- **Shuffle and Sort Phase**: The system groups all the key-value pairs by the key (i.e., word), so all occurrences of the same word are collected together.
- **Reduce Phase**: The reducer processes the grouped data, summing the values for each key (word) to get the final frequency count.

Advantages:

- **Scalability**: The MapReduce model can process **huge datasets** because the work is split into smaller, independent tasks (map phase) and then aggregated (reduce phase). This makes it efficient for large-scale distributed systems (e.g., Hadoop, Spark).
- **Parallel Processing**: Each map function works independently on different parts of the dataset, and reduce operations can also be parallelized. This makes the algorithm efficient even with vast amounts of data.

b) Aim: Implement a Map Reduce program that processes a weather dataset.

Description:

MapReduce for Weather Data is a way to process large weather datasets efficiently by breaking down the task into smaller, manageable units. This approach uses the MapReduce programming model, which is often employed in distributed computing frameworks like Hadoop and Apache Spark. In the context of weather data, it can be used to perform operations such as calculating average temperatures or aggregating weather data.
The goal of the MapReduce for Weather Data program is to process a dataset of weather records and calculate useful metrics, like the average high and low temperatures for each date.

Where each record represents:
- A date (2023-05-01)
- A high temperature (72)
- A low temperature (58)
- A weather condition (Clear)

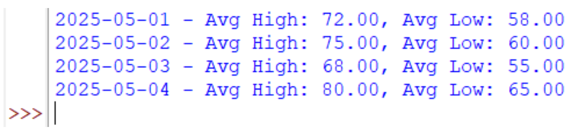
In the MapReduce model, you would calculate the average high and low temperatures for each date.
- MapReduce Programming Model Breakdown
- MapReduce involves two main steps:

Map Step (Mapping Phase):
- The input data is divided into chunks and processed in parallel by mappers.
- In the case of weather data, each line of the dataset would be processed to extract the necessary information.

Reduce Step (Reducing Phase):
  - The results of the map phase are grouped by key (in this case, the date), and then reducers aggregate the results (in this case, computing the average high and low temperatures for each date).

Code:
```
from collections import defaultdict

# Map function: Processes weather data and emits date with temperature data
def map_weather_function(record):
    date, high, low, _ = record.split(",")
    high = int(high)
    low = int(low)
    yield (date, (high, low))

# Reduce function: Computes the average temperature for a given date
def reduce_weather_function(date, temp_data):
    total_high = total_low = 0
    count = 0
    for high, low in temp_data:
        total_high += high
        total_low += low
        count += 1
    avg_high = total_high / count
    avg_low = total_low / count
    return (avg_high, avg_low)

# Driver function to simulate MapReduce on weather data
def weather_map_reduce(weather_data):
    # Map step
    intermediate_data = defaultdict(list)
    for record in weather_data:
        for date, temps in map_weather_function(record):
            intermediate_data[date].append(temps)

    # Reduce step
    result = {}
    for date, temp_data in intermediate_data.items():
        result[date] = reduce_weather_function(date, temp_data)

    return result


# Example weather data (CSV format as a list of strings)
weather_data = [
    "2025-05-01,72,58,Clear",
    "2025-05-02,75,60,Cloudy",
    "2025-05-03,68,55,Rain",
    "2025-05-04,80,65,Clear",
]

# Run the MapReduce simulation
avg_temperatures = weather_map_reduce(weather_data)

# Output result
for date, (avg_high, avg_low) in avg_temperatures.items():
    print(f"{date} - Avg High: {avg_high:.2f}, Avg Low: {avg_low:.2f}")
```

<u>Output</u>:



```
2025-05-01 - Avg High: 72.00, Avg Low: 58.00
2025-05-02 - Avg High: 75.00, Avg Low: 60.00
2025-05-03 - Avg High: 68.00, Avg Low: 55.00
2025-05-04 - Avg High: 80.00, Avg Low: 65.00
>>> |
```

Learnings:

Explanation of the Process

1. Map Phase:
    - The map function processes each weather record, extracting the date, high, and low temperatures, and emitting key-value pairs like (date, (high, low)).
    - These pairs are grouped by the date in intermediate_data.
2. Reduce Phase:
    - The reducer processes the grouped data for each date and computes the average of the high and low temperatures.
    - The averages are stored in the result dictionary.

Why Use MapReduce for Weather Data?

- Scalability: MapReduce can handle very large datasets efficiently by dividing the work into small, independent tasks (map phase) and aggregating the results (reduce phase). If you had millions of records of weather data, this approach would scale to handle the data distributed across a cluster.
- Parallel Processing: The map function processes each record independently, and the reduce function aggregates the results in parallel. This makes it efficient for distributed environments like Hadoop or Spark.
- Distributed Computing: In distributed environments, each mapper runs on different nodes, which allows for efficient parallel computation, especially when working with big data.

Summary of the Weather Data MapReduce Program:

1. Input: A set of weather records, each with a date, high and low temperature, and weather condition.
2. Map Phase: Extracts the date and temperatures, emitting key-value pairs for each record.
3. Reduce Phase: Computes the average high and low temperatures for each date.
4. Output: A dictionary containing the date as the key and a tuple

Practical :- 09

Aim: Implement an application that stores big data in MongoDB and manipulate it using R/Python

Description: Procedure

1. **Install MongoDB**:
   Download and install MongoDB from the official site and start the MongoDB server (usually on localhost:27017).
2. **Install PyMongo**:
   Use pip install pymongo to install the MongoDB driver for Python.
3. **Connect to MongoDB**:
   Use pymongo.MongoClient() to connect to the MongoDB server.
4. **Create Database & Collection**:
   Access or create a database and a collection (like a table).
5. **Insert Data**:
   Use insert_one() or insert_many() to add documents (records).
6. **Query Data**:
   Use find() or find_one() to retrieve data with or without conditions.
7. **Update Data**:
   Use update_one() or update_many() to modify existing documents.
8. **Delete Data**:
   Use delete_one() or delete_many() to remove documents.
9. **List Databases & Collections**:
   Use list_database_names() and list_collection_names() to inspect your data.
10. **Close Connection** (Optional):
    Use client.close() when you're done.

MongoDB + PyMongo is a powerful combination for building flexible, scalable, and efficient data-driven applications using Python.

Code 1:
```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mybigdata"]
print(myclient.list_database_names())
```

Code 2:
```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mybigdata"]
mycol=mydb["student"]
print(mydb.list_collection_names())
```

Code 3:
```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mybigdata"]
mycol=mydb["student"]
mydict={"name":"Beena", "address":"Mumbai"}
x=mycol.ix=mycol.insert_one(mydict) # insert_one(containing the name(s) and value(s) of each field
```

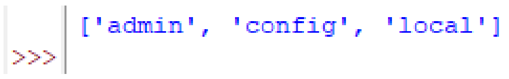
Code 4:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mybigdataNK"]
print(myclient.list_database_names())
mycol=mydb["student"]
print(mydb.list_collection_names())
mydict={"name":"Ninad", "address":"Mumbai"}
x=mycol.insert_one(mydict) # insert_one(containing the name(s) and value(s) of each field
mylist=[{"name":"Vighnesh", "address":"Mumbai"}, {"name":"Sarthak", "address":"Mumbai"},
{"name":"Nidhi", "address":"Pune"}, {"name":"Komal", "address":"Pune"},]
x=mycol.insert_many(mylist)
```

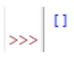
CMD COMMANDS:

```
> show dbs
> use mybigdataNK
> show collections
>  db.student.findOne()
> db.student.count()
```

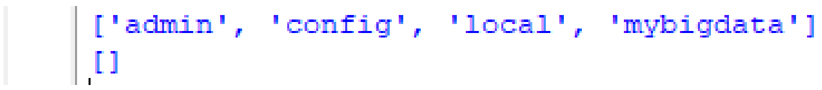
Output:



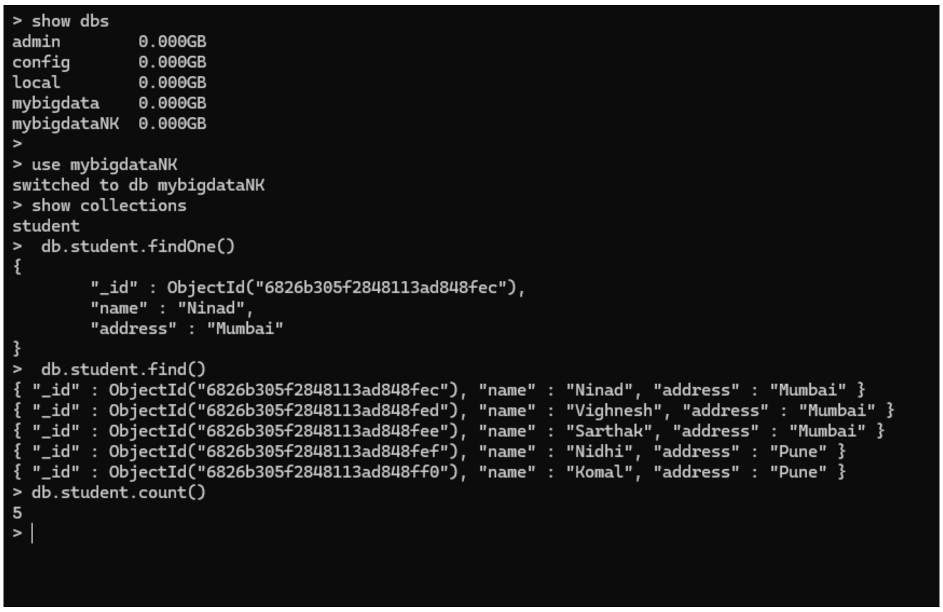
```
['admin', 'config', 'local']
>>>
```



```
[]
>>>
```



```
['admin', 'config', 'local', 'mybigdata']
[]
```



```
> show dbs
admin        0.000GB
config       0.000GB
local        0.000GB
mybigdata    0.000GB
mybigdataNK  0.000GB
>
> use mybigdataNK
switched to db mybigdataNK
> show collections
student
>  db.student.findOne()
{
        "_id" : ObjectId("6826b305f2848113ad848fec"),
        "name" : "Ninad",
        "address" : "Mumbai"
}
>  db.student.find()
{ "_id" : ObjectId("6826b305f2848113ad848fec"), "name" : "Ninad", "address" : "Mumbai" }
{ "_id" : ObjectId("6826b305f2848113ad848fed"), "name" : "Vighnesh", "address" : "Mumbai" }
{ "_id" : ObjectId("6826b305f2848113ad848fee"), "name" : "Sarthak", "address" : "Mumbai" }
{ "_id" : ObjectId("6826b305f2848113ad848fef"), "name" : "Nidhi", "address" : "Pune" }
{ "_id" : ObjectId("6826b305f2848113ad848ff0"), "name" : "Komal", "address" : "Pune" }
> db.student.count()
5
> |
```

Learnings:

- These Python scripts create MongoDB databases and collections, and insert data using pymongo.
- The CMD commands allow you to inspect the data directly using the MongoDB shell.

MongoDB

- A NoSQL, document-oriented database.
- Designed for flexibility, scalability, and high performance.
- Stores data in JSON-like documents (called BSON – Binary JSON), not rows and columns like relational databases (e.g., MySQL, PostgreSQL).

  Key Features:
  - Schema-less: You don't need to define a strict schema in advance.
  - Document-based: Each document is a key-value pair structure (like a Python dictionary).
  - Horizontal scalability using sharding (splitting data across multiple machines).
  - Rich query language: Supports CRUD operations, filtering, aggregation, indexing, etcOften used in web applications, real-time analytics, IoT, and big data.

PyMongo
The official Python driver for interacting with MongoDB.

- Allows Python applications to connect, read, write, and query MongoDB databases.
- Developed and maintained by MongoDB, Inc.

  Key Features:
  - Full support for all MongoDB operations (insert, update, delete, query, etc.).
  - Provides an intuitive way to work with MongoDB using Python dictionaries.
  - Integrates well with other Python tools and frameworks.