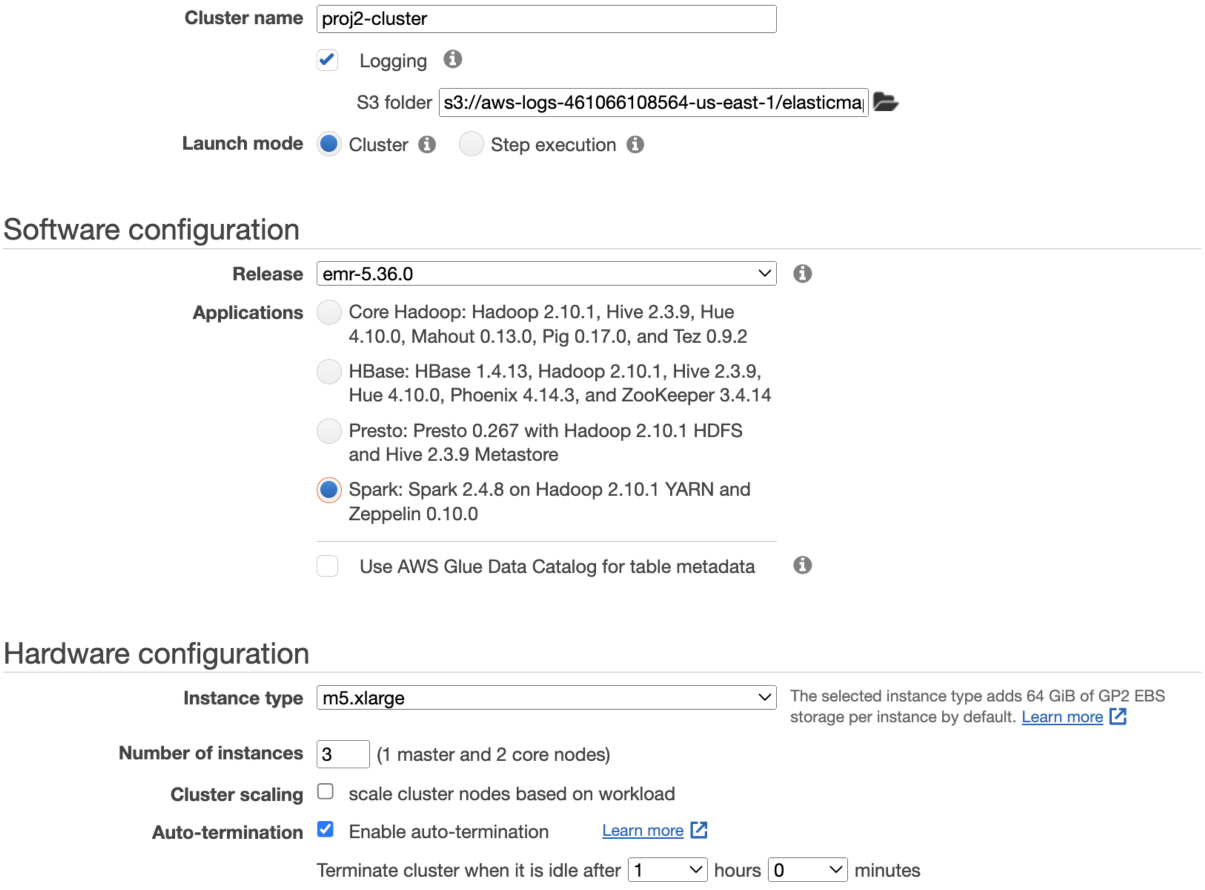
Alif Rahi - CSCI 381 ***Project 2 : Hadoop and Spark***

<u>**Task 1 & 2:**</u> [Launch a cluster of virtual machines in a cloud environment]

Amazon EMR is a managed big data processing service offered by Amazon Web Services. In EMR, the **master** and **slave** roles refer to different types of instances that are used in an EMR cluster. To create an Amazon EMR cluster, you can follow these steps:

1. Sign in to your aws Console and go to the Amazon emr service page.
2. Click on the "Create cluster" button.
3. Choose the appropriate key pair if you need SSH access to the cluster instances.
4. In the configuration section, choose the applications and versions you want to install on the cluster. For this project you should select **Apache Spark**.



Cluster name proj2-cluster

☑ Logging

S3 folder s3://aws-logs-461066108564-us-east-1/elasticma|

Launch mode ● Cluster ○ Step execution

Software configuration

Release emr-5.36.0

Applications

○ Core Hadoop: Hadoop 2.10.1, Hive 2.3.9, Hue 4.10.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2

○ HBase: HBase 1.4.13, Hadoop 2.10.1, Hive 2.3.9, Hue 4.10.0, Phoenix 4.14.3, and ZooKeeper 3.4.14

○ Presto: Presto 0.267 with Hadoop 2.10.1 HDFS and Hive 2.3.9 Metastore

● Spark: Spark 2.4.8 on Hadoop 2.10.1 YARN and Zeppelin 0.10.0

□ Use AWS Glue Data Catalog for table metadata

Hardware configuration

Instance type m5.xlarge — The selected instance type adds 64 GiB of GP2 EBS storage per instance by default. Learn more

Number of instances 3 (1 master and 2 core nodes)

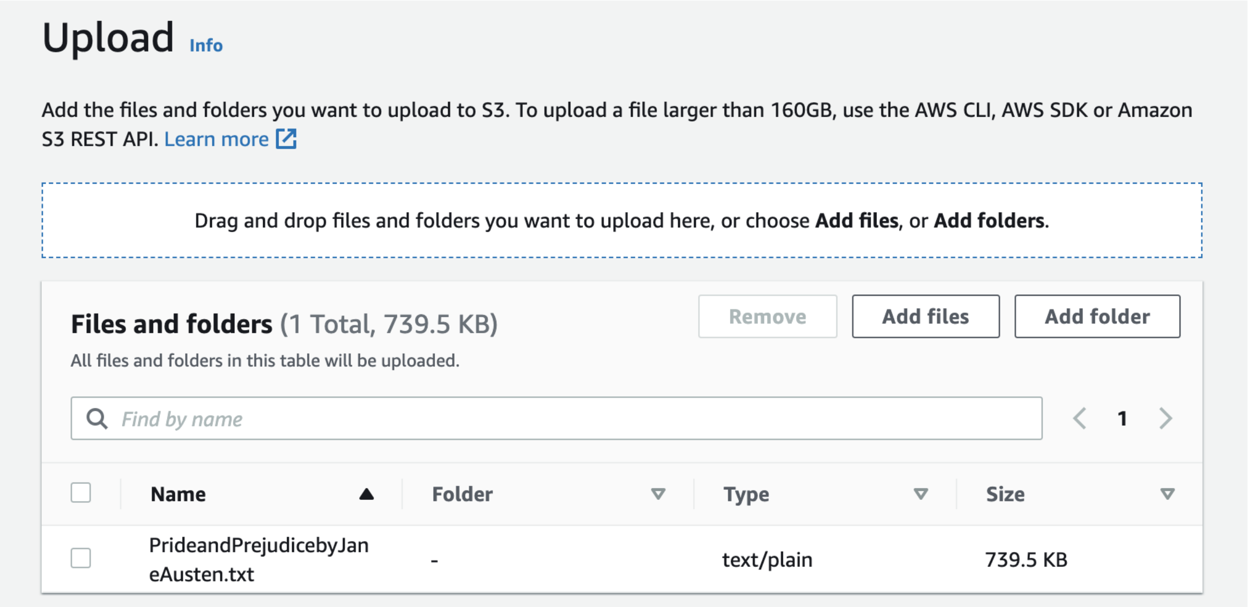
Cluster scaling □ scale cluster nodes based on workload

Auto-termination ☑ Enable auto-termination Learn more

Terminate cluster when it is idle after 1 hours 0 minutes

Task 3: [Download txt file and save it to the HDFS cluster]

Amazon EMR provides several ways to get data onto a cluster. The most common way is to upload the data to **Amazon S3** and use the built-in features of Amazon EMR to load the data onto your cluster.



Upload Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more

Drag and drop files and folders you want to upload here, or choose **Add files**, or **Add folders**.

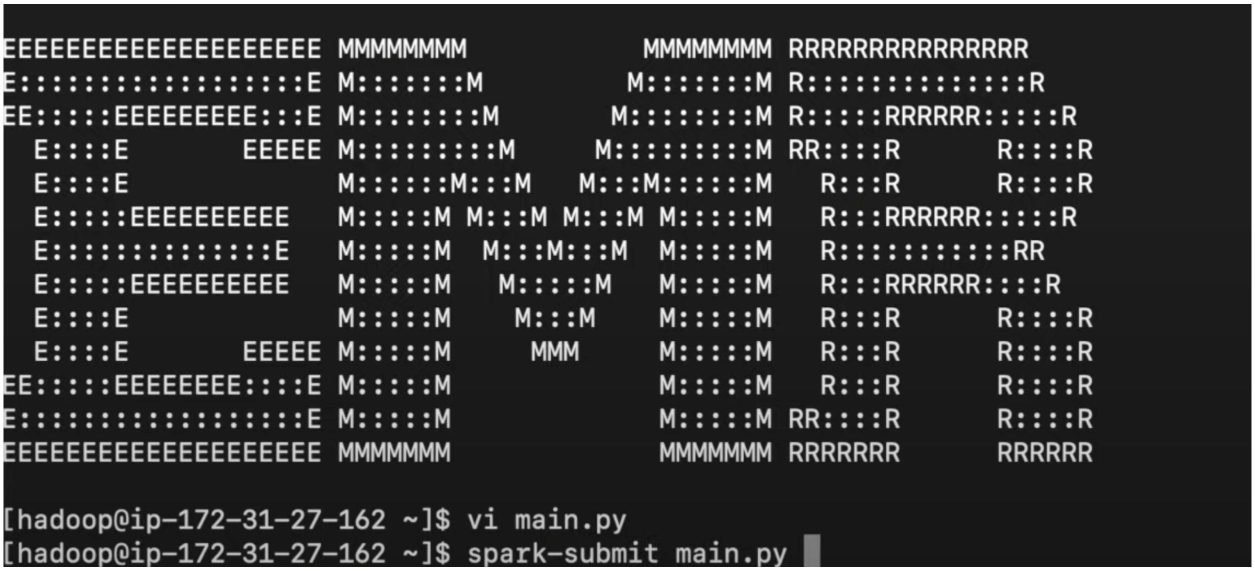
Files and folders (1 Total, 739.5 KB)

All files and folders in this table will be uploaded.

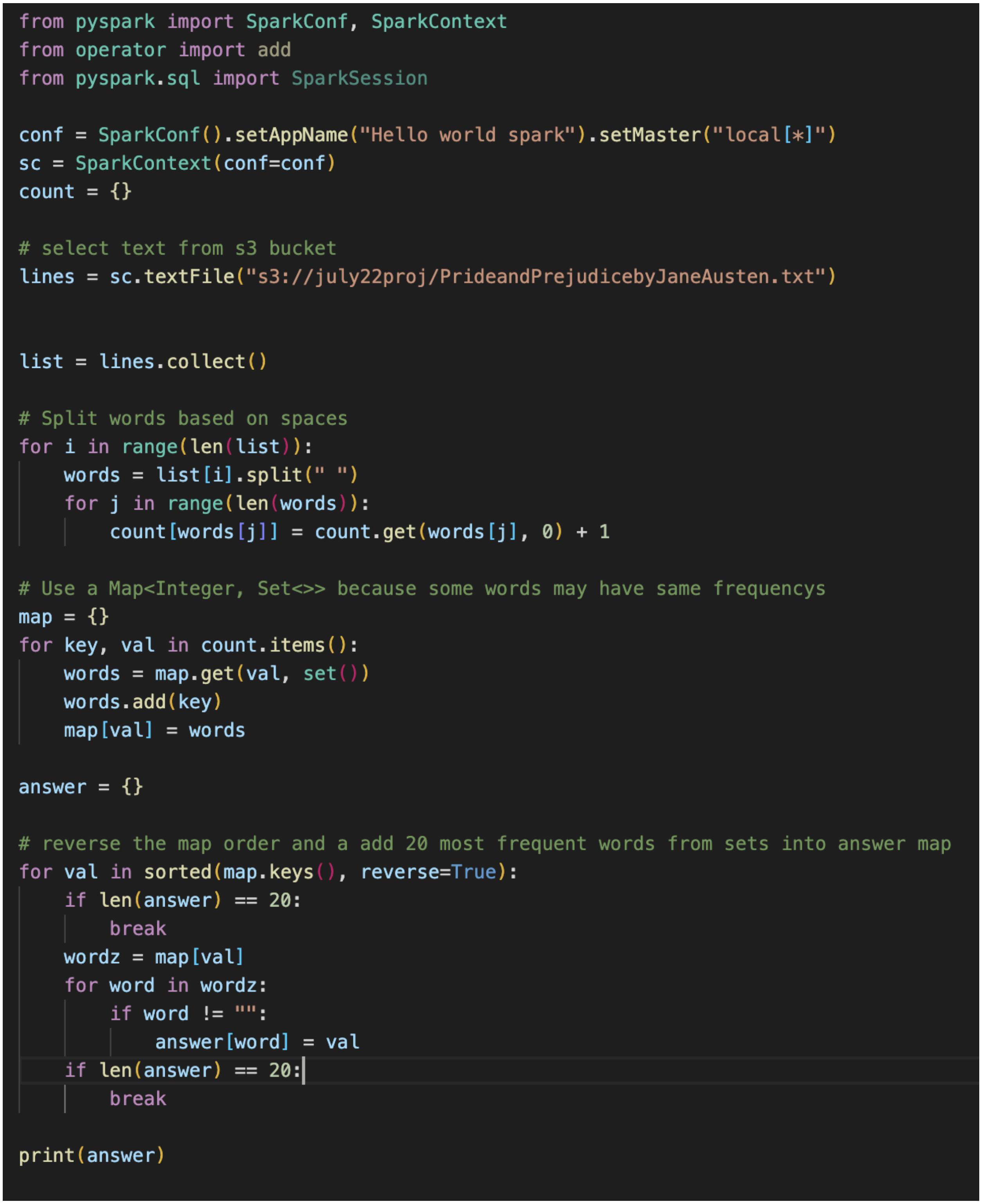
□	Name ▲	Folder ▽	Type ▽	Size ▽
□	PrideandPrejudicebyJaneAusten.txt	-	text/plain	739.5 KB

Task 4 & 5: [20 most frequent words in PrideandPrejudicebyJaneAusten.txt]

Once you have created the Spark application, you can submit it to your Spark cluster using the spark-submit command after logging into your EMR instance

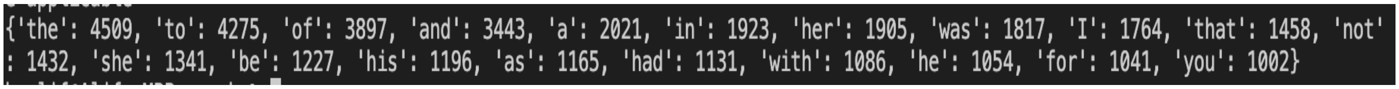


```
EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-27-162 ~]$ vi main.py
[hadoop@ip-172-31-27-162 ~]$ spark-submit main.py
```



```python
from pyspark import SparkConf, SparkContext
from operator import add
from pyspark.sql import SparkSession

conf = SparkConf().setAppName("Hello world spark").setMaster("local[*]")
sc = SparkContext(conf=conf)
count = {}

# select text from s3 bucket
lines = sc.textFile("s3://july22proj/PrideandPrejudicebyJaneAusten.txt")


list = lines.collect()

# Split words based on spaces
for i in range(len(list)):
    words = list[i].split(" ")
    for j in range(len(words)):
        count[words[j]] = count.get(words[j], 0) + 1

# Use a Map<Integer, Set<>> because some words may have same frequencys
map = {}
for key, val in count.items():
    words = map.get(val, set())
    words.add(key)
    map[val] = words

answer = {}

# reverse the map order and a add 20 most frequent words from sets into answer map
for val in sorted(map.keys(), reverse=True):
    if len(answer) == 20:
        break
    wordz = map[val]
    for word in wordz:
        if word != "":
            answer[word] = val
    if len(answer) == 20:
        break

print(answer)
```

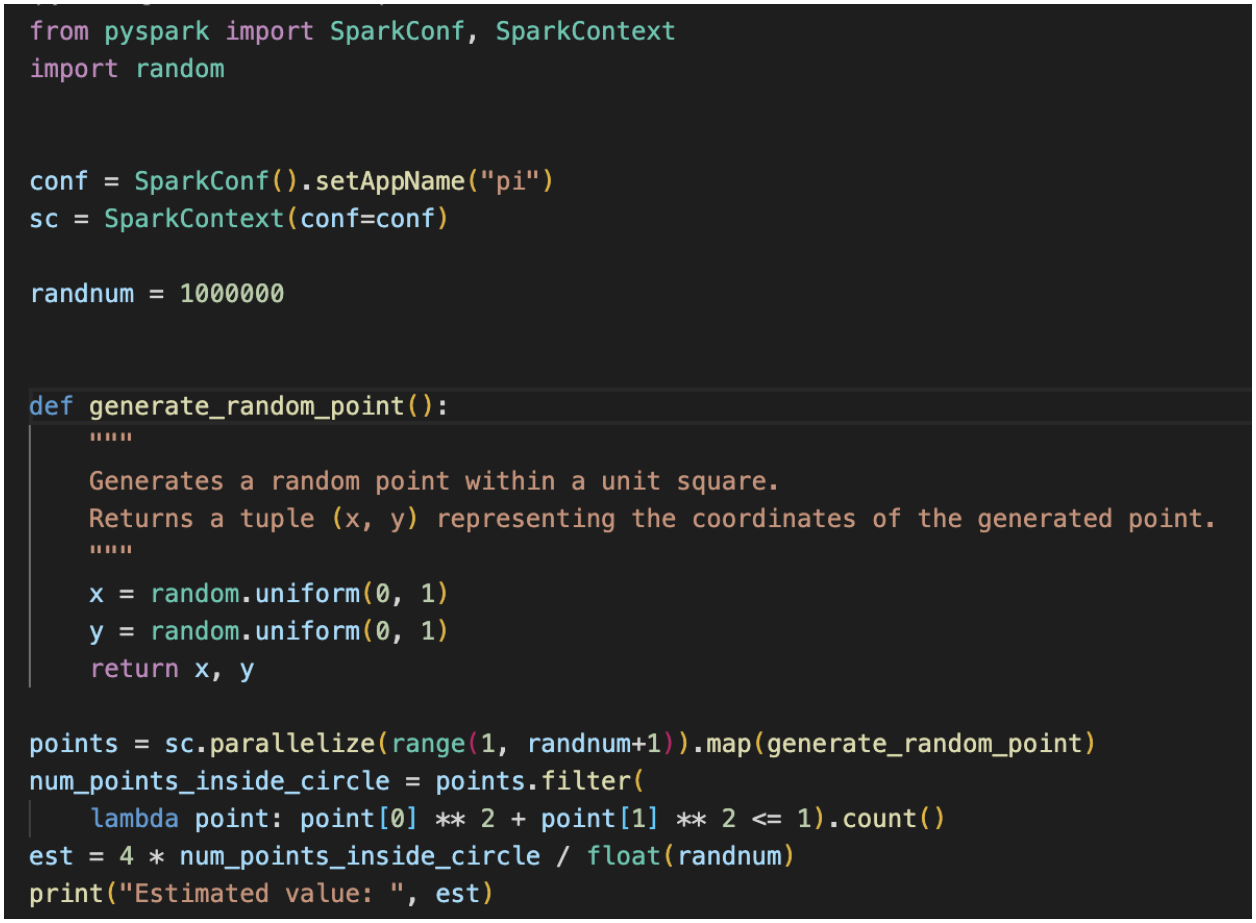
Output



```
{'the': 4509, 'to': 4275, 'of': 3897, 'and': 3443, 'a': 2021, 'in': 1923, 'her': 1905, 'was': 1817, 'I': 1764, 'that': 1458, 'not'
: 1432, 'she': 1341, 'be': 1227, 'his': 1196, 'as': 1165, 'had': 1131, 'with': 1086, 'he': 1054, 'for': 1041, 'you': 1002}
```

Task 6:

Monte Carlo refers to simulation technique that uses **random** numbers to estimate unknown quantities or analyze complex systems.



```python
from pyspark import SparkConf, SparkContext
import random


conf = SparkConf().setAppName("pi")
sc = SparkContext(conf=conf)

randnum = 1000000


def generate_random_point():
    """
    Generates a random point within a unit square.
    Returns a tuple (x, y) representing the coordinates of the generated point.
    """
    x = random.uniform(0, 1)
    y = random.uniform(0, 1)
    return x, y

points = sc.parallelize(range(1, randnum+1)).map(generate_random_point)
num_points_inside_circle = points.filter(
    lambda point: point[0] ** 2 + point[1] ** 2 <= 1).count()
est = 4 * num_points_inside_circle / float(randnum)
print("Estimated value: ", est)
```

- You can call this **random.uniform** function multiple times to generate a desired number of random points within the unit square, as shown in the example of estimating π using Monte Carlo methods.
- The **sc.parallelize** method is used to create a distributed collection of data, called an RDD from a local collection of data in the driver program.