# Big Data Analytics
# Exercise Sheet 2 Solution

**Abdul Rehman Liaqat, 271336**

## Exercise 1:

The *Init(args)* method of MPI, constructs all global and internal variables of MPI. Each process will be given a unique number/rank and a communicator function will be formed inside which will be responsible for all the sending and receiving commands from other processes and from the main process.
*Finalize( )* methods cleans up the MPI environment that means it will remove any connection(communicator) with other process in all processes and no MPI interaction can be made after this function call.
*COMM_WORLD( )* this is the global communicator function which is designed by MPI for us as an abstraction to each local processes communication function. There are two main methods to each COMM_WORLD(): Get_size() and Get_rank(). Get_size() will return the number of processes being handled by COMM_WORLD() function, and the Get_rank() function will return the rank of the local worker. This way it can be said that COMM_WORLD() is the communication port for each process. It handles all kind of send and receiving information and communication.

For example:

1- We commanded an MPI execution on 4 processes/cores by using “mpiexec” command.
2- First *Init(args)* will work which will initialize the function variables and local copy of COMM_WORLD().
3- Next up we can make this process transfer information/data and communicate with any of other 3 processes through local copy of COMM_WORLD() function.
4- Finally after all tasks are done. *Finalize()* will remove all the variables and local communicators in each processes separately.

## Exercise 2:

The possible problem related to blocking point-to-point communication in the following code is of possibility of deadlock.
In MPI each kind of communication can be boiled down to a basic *Send()* and *Receive()* function. When a process say A wants to send a message to another process say B, it will pack the message/package in a data structure and route through the network. B will respond if it needs the package and then the message will be transmitted to B. B will acknowledge the reception of message. Note that during this whole process; transmitting message, receiving message, waiting to receive the whole message, sending acknowledgement and waiting to

receive acknowledgement both processes remains in blocking mode. Which means that A will not receive anything and B will not transmit anything. As shown below:

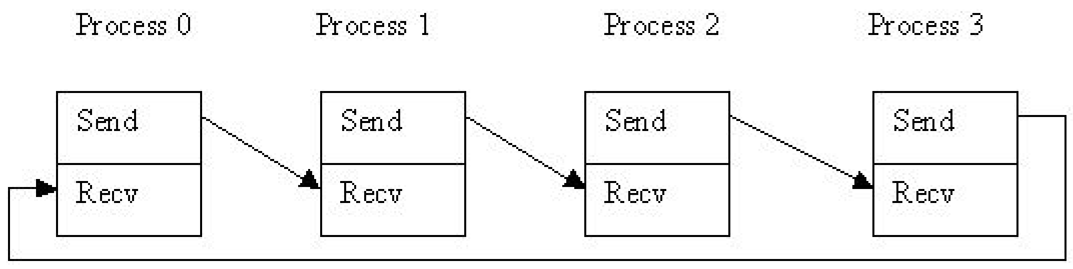
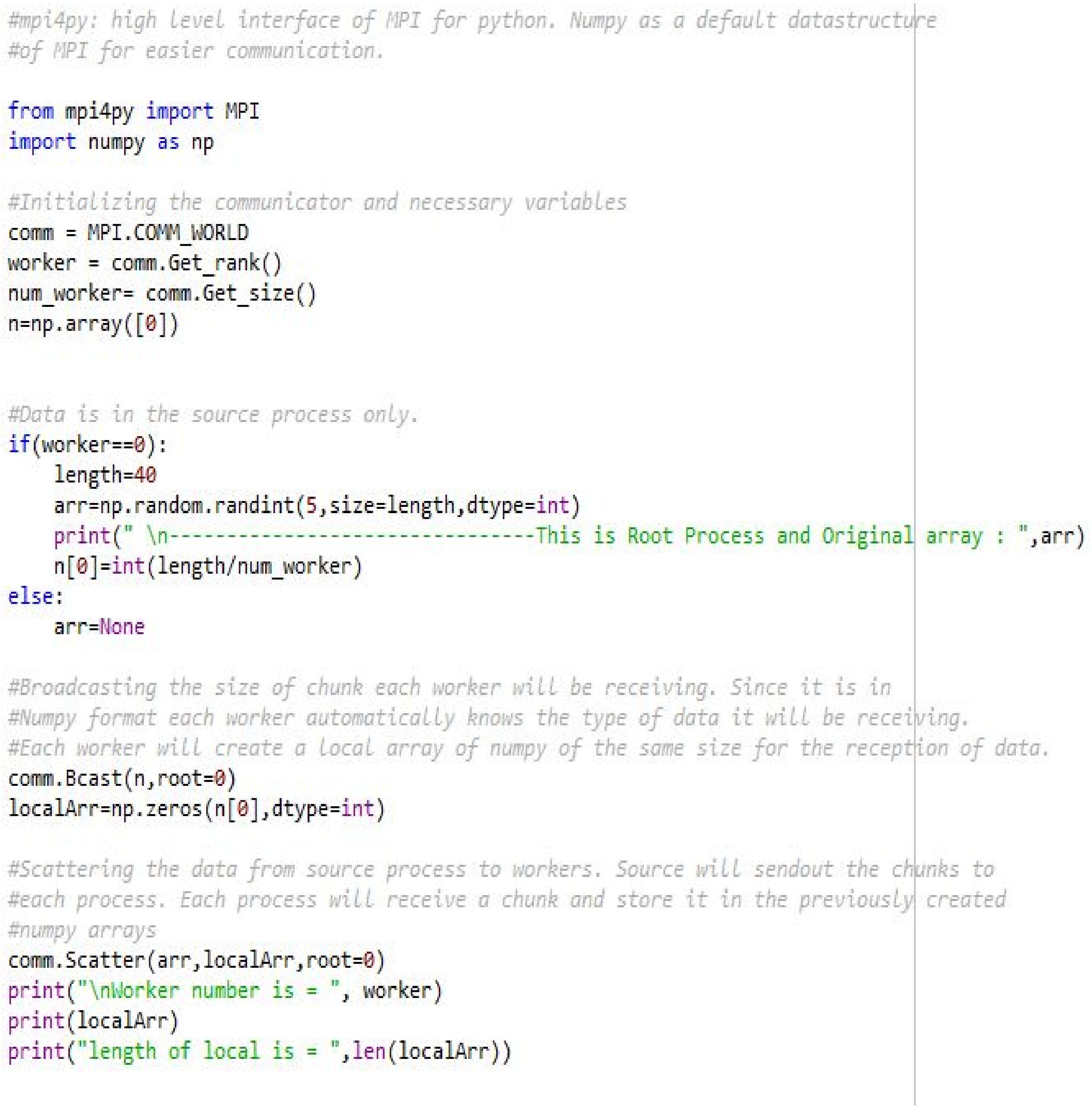


Figure 2. Dependency cycle with four processes.

In a distributed environment especially where cores and physically located at distance and network can suffer from latency, it is possible that the system given in the code suffer from deadlock. Each process is in blocking mode to receive or send. This way no one receive the message and the whole communication is stuck in deadlock.

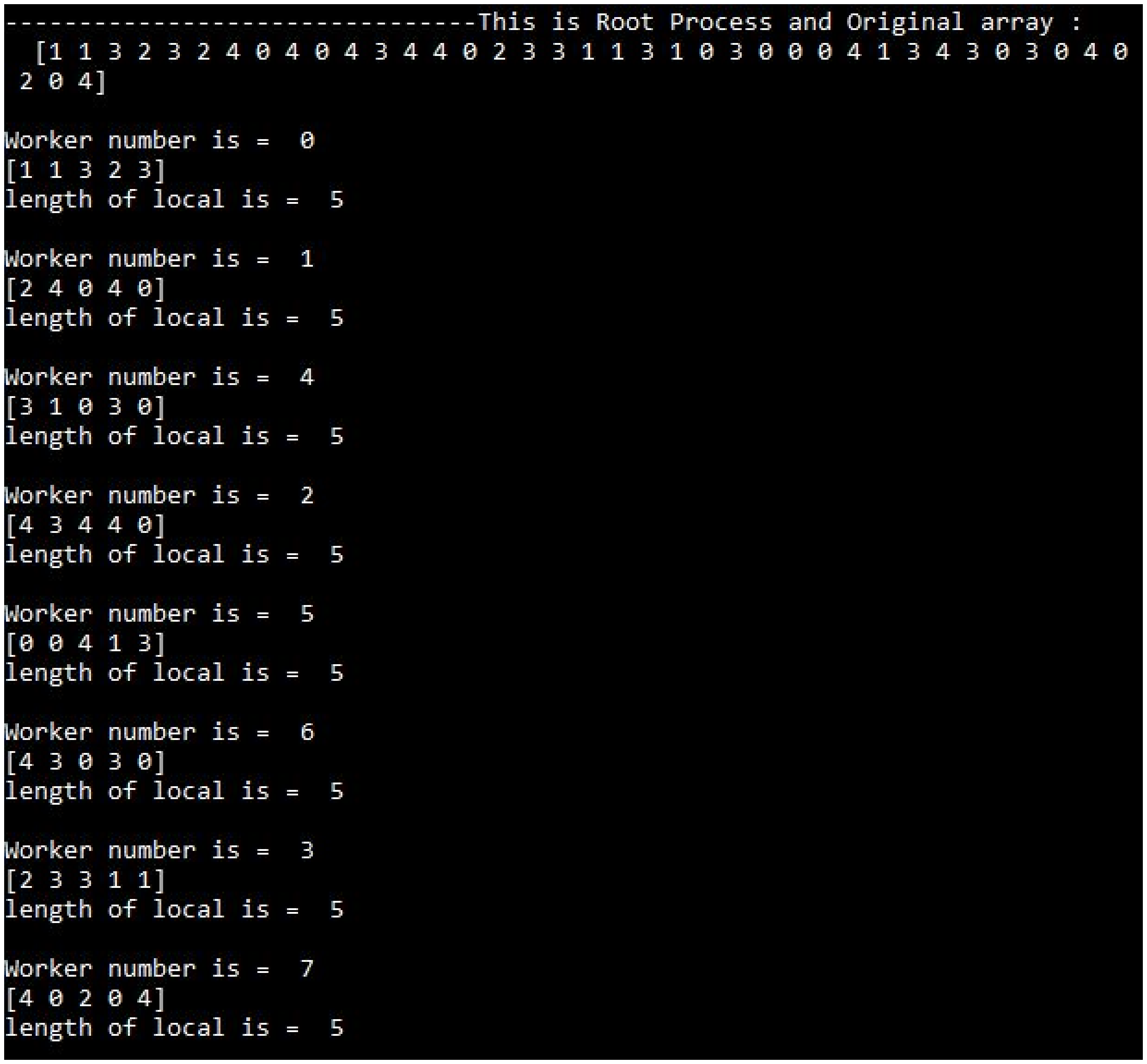
One solution is to change it from blocking to non-blocking point-to-point communication methods. This way after sending the message a process will get to work and not wait for acknowledgement. This is the same in case of reception.

**Exercise 3a:**



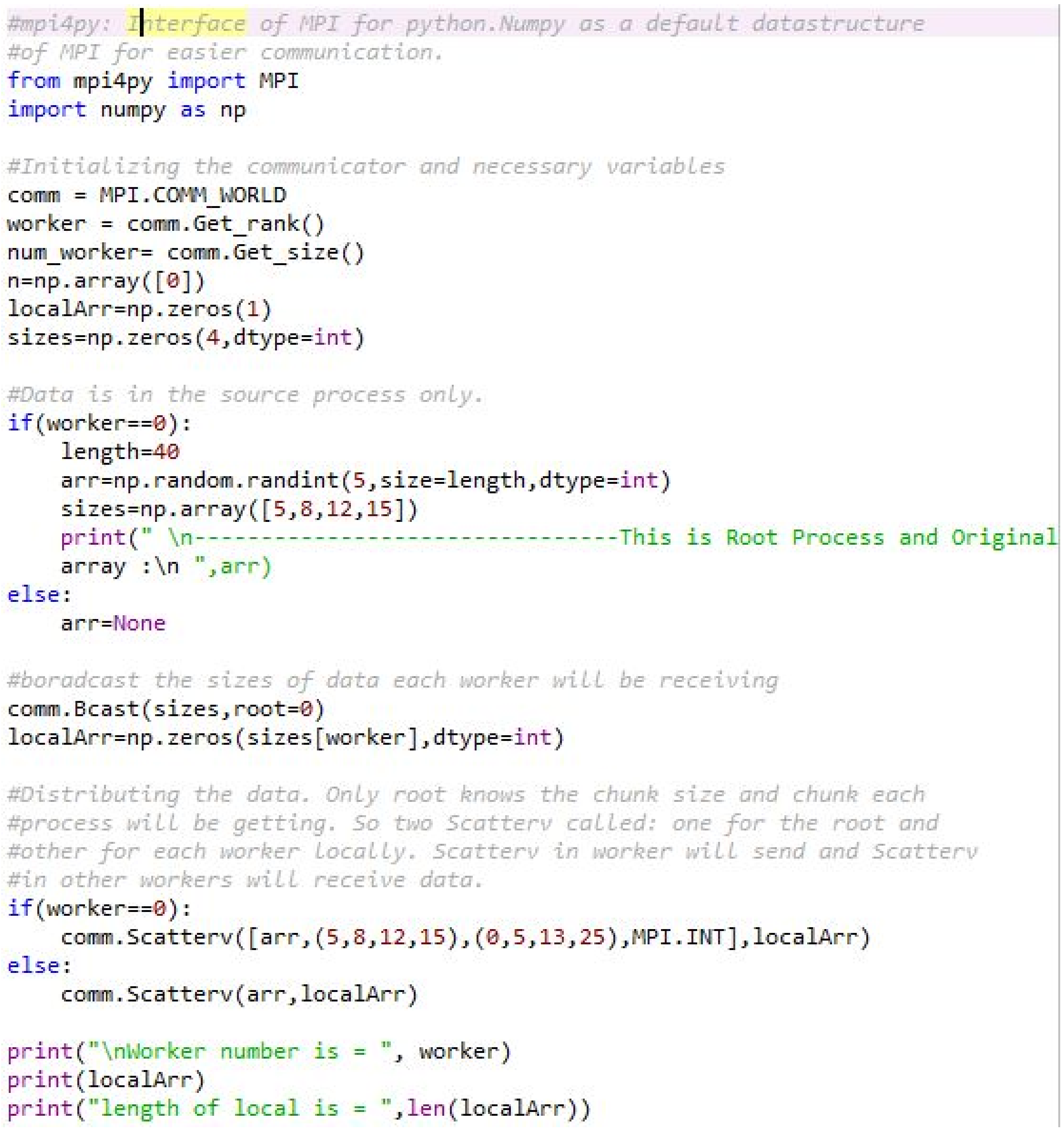
```python
#mpi4py: high level interface of MPI for python. Numpy as a default datastructure
#of MPI for easier communication.

from mpi4py import MPI
import numpy as np

#Initializing the communicator and necessary variables
comm = MPI.COMM_WORLD
worker = comm.Get_rank()
num_worker= comm.Get_size()
n=np.array([0])


#Data is in the source process only.
if(worker==0):
    length=40
    arr=np.random.randint(5,size=length,dtype=int)
    print(" \n-------------------------------This is Root Process and Original array : ",arr)
    n[0]=int(length/num_worker)
else:
    arr=None

#Broadcasting the size of chunk each worker will be receiving. Since it is in
#Numpy format each worker automatically knows the type of data it will be receiving.
#Each worker will create a local array of numpy of the same size for the reception of data.
comm.Bcast(n,root=0)
localArr=np.zeros(n[0],dtype=int)

#Scattering the data from source process to workers. Source will sendout the chunks to
#each process. Each process will receive a chunk and store it in the previously created
#numpy arrays
comm.Scatter(arr,localArr,root=0)
print("\nWorker number is = ", worker)
print(localArr)
print("length of local is = ",len(localArr))
```

**Output:**



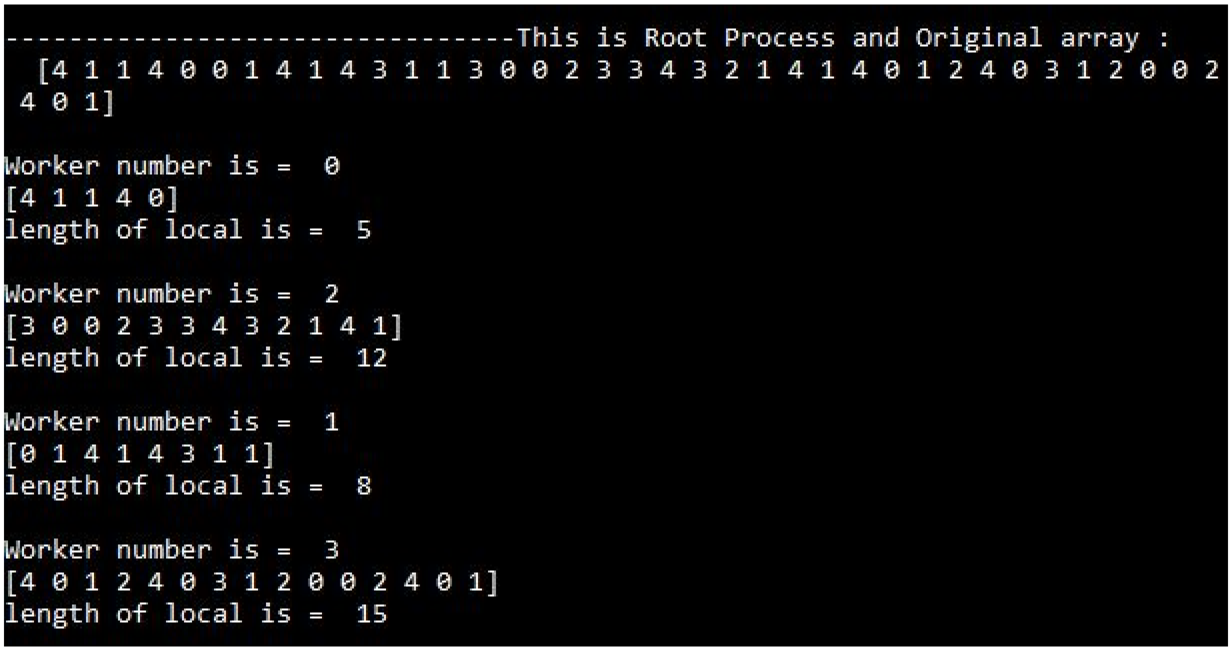
```
--------------------------------This is Root Process and Original array :
  [1 1 3 2 3 2 4 0 4 0 4 3 4 4 0 2 3 3 1 1 3 1 0 3 0 0 0 4 1 3 4 3 0 3 0 4 0
 2 0 4]

Worker number is =  0
[1 1 3 2 3]
length of local is =  5

Worker number is =  1
[2 4 0 4 0]
length of local is =  5

Worker number is =  4
[3 1 0 3 0]
length of local is =  5

Worker number is =  2
[4 3 4 4 0]
length of local is =  5

Worker number is =  5
[0 0 4 1 3]
length of local is =  5

Worker number is =  6
[4 3 0 3 0]
length of local is =  5

Worker number is =  3
[2 3 3 1 1]
length of local is =  5

Worker number is =  7
[4 0 2 0 4]
length of local is =  5
```

**Exercise 3b:**



```
#mpi4py: Interface of MPI for python.Numpy as a default datastructure
#of MPI for easier communication.
from mpi4py import MPI
import numpy as np

#Initializing the communicator and necessary variables
comm = MPI.COMM_WORLD
worker = comm.Get_rank()
num_worker= comm.Get_size()
n=np.array([0])
localArr=np.zeros(1)
sizes=np.zeros(4,dtype=int)

#Data is in the source process only.
if(worker==0):
    length=40
    arr=np.random.randint(5,size=length,dtype=int)
    sizes=np.array([5,8,12,15])
    print(" \n--------------------------------This is Root Process and Original
    array :\n ",arr)
else:
    arr=None

#boradcast the sizes of data each worker will be receiving
comm.Bcast(sizes,root=0)
localArr=np.zeros(sizes[worker],dtype=int)

#Distributing the data. Only root knows the chunk size and chunk each
#process will be getting. So two Scatterv called: one for the root and
#other for each worker locally. Scatterv in worker will send and Scatterv
#in other workers will receive data.
if(worker==0):
    comm.Scatterv([arr,(5,8,12,15),(0,5,13,25),MPI.INT],localArr)
else:
    comm.Scatterv(arr,localArr)

print("\nWorker number is = ", worker)
print(localArr)
print("length of local is = ",len(localArr))
```

**Output**:



```
-------------------------------This is Root Process and Original array :
  [4 1 1 4 0 0 1 4 1 4 3 1 1 3 0 0 2 3 3 4 3 2 1 4 1 4 0 1 2 4 0 3 1 2 0 0 2
 4 0 1]

Worker number is =  0
[4 1 1 4 0]
length of local is =  5

Worker number is =  2
[3 0 0 2 3 3 4 3 2 1 4 1]
length of local is =  12

Worker number is =  1
[0 1 4 1 4 3 1 1]
length of local is =  8

Worker number is =  3
[4 0 1 2 4 0 3 1 2 0 0 2 4 0 1]
length of local is =  15
```