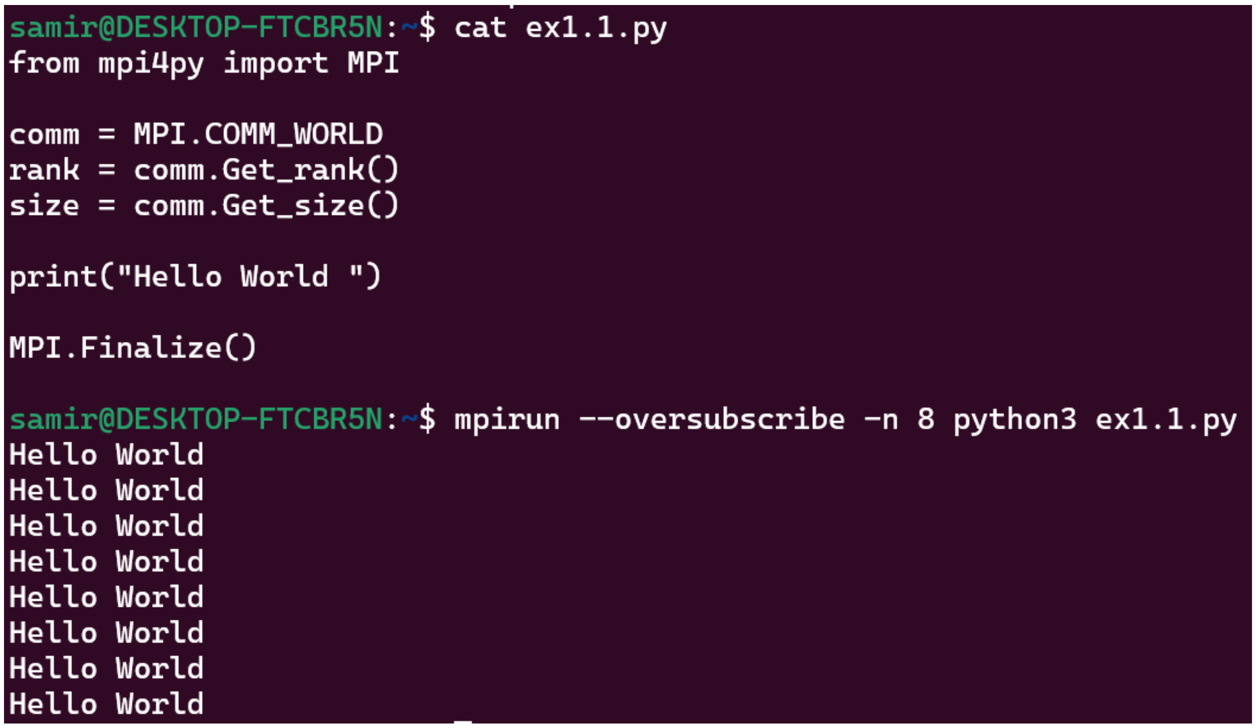
Samir JABBAR

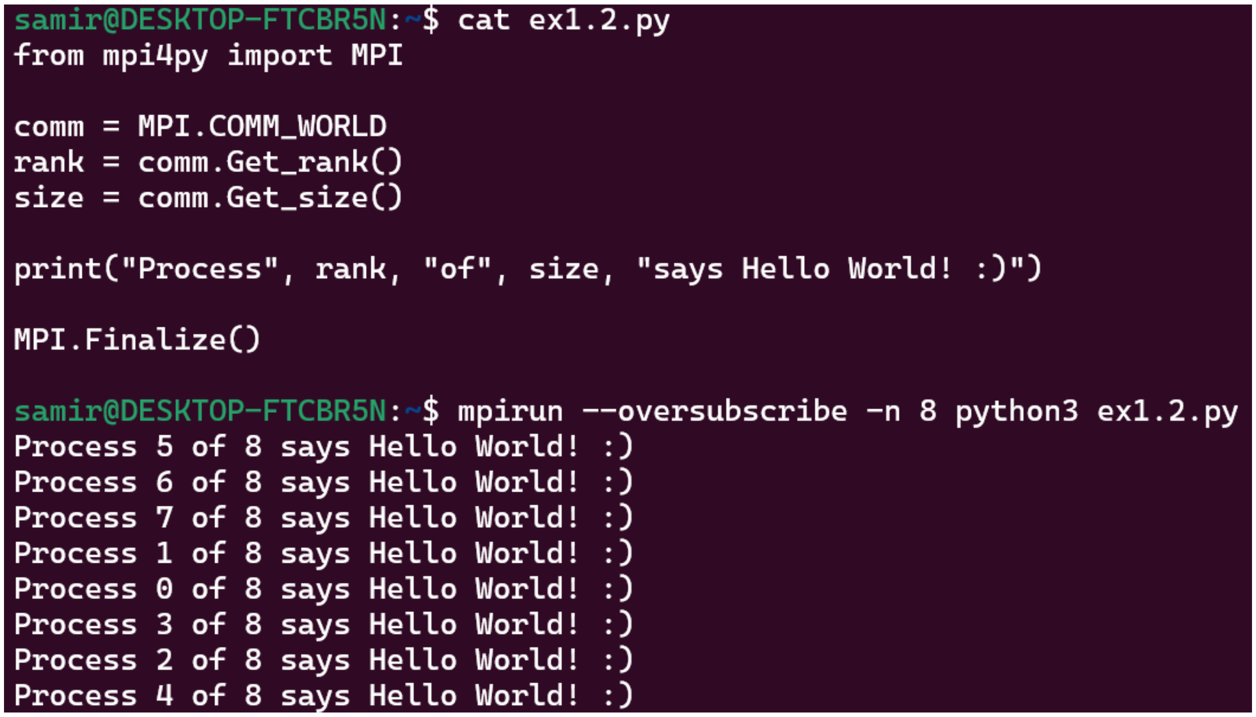
Exercice 1

Q1



```
samir@DESKTOP-FTCBR5N:~$ cat ex1.1.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

print("Hello World ")

MPI.Finalize()

samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex1.1.py
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

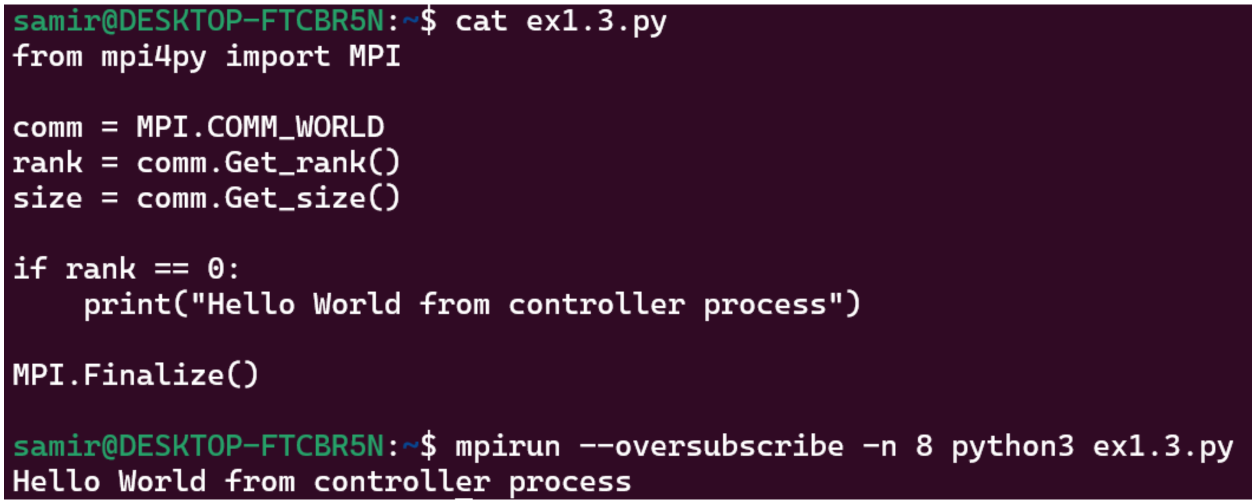
Q2



```
samir@DESKTOP-FTCBR5N:~$ cat ex1.2.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

print("Process", rank, "of", size, "says Hello World! :)")

MPI.Finalize()

samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex1.2.py
Process 5 of 8 says Hello World! :)
Process 6 of 8 says Hello World! :)
Process 7 of 8 says Hello World! :)
Process 1 of 8 says Hello World! :)
Process 0 of 8 says Hello World! :)
Process 3 of 8 says Hello World! :)
Process 2 of 8 says Hello World! :)
Process 4 of 8 says Hello World! :)
```

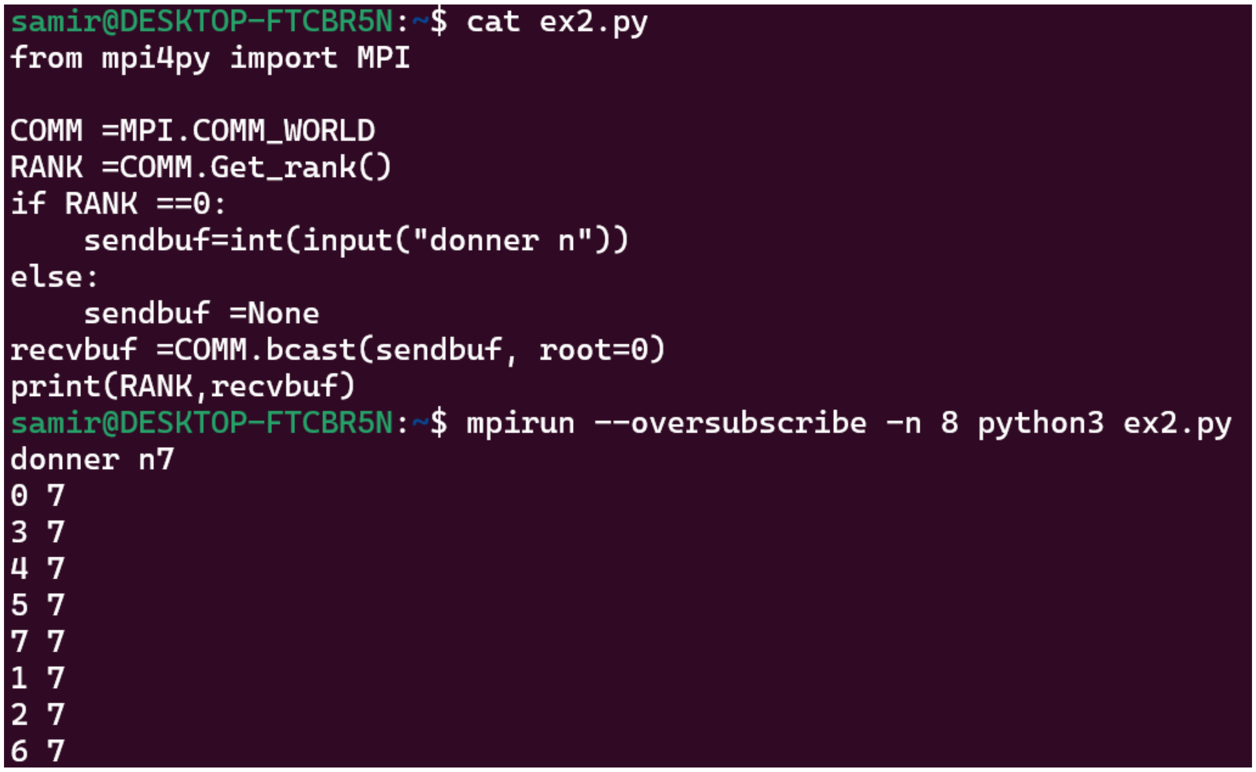
Samir JABBAR

Q3



```
samir@DESKTOP-FTCBR5N:~$ cat ex1.3.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if rank == 0:
    print("Hello World from controller process")

MPI.Finalize()

samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex1.3.py
Hello World from controller process
```

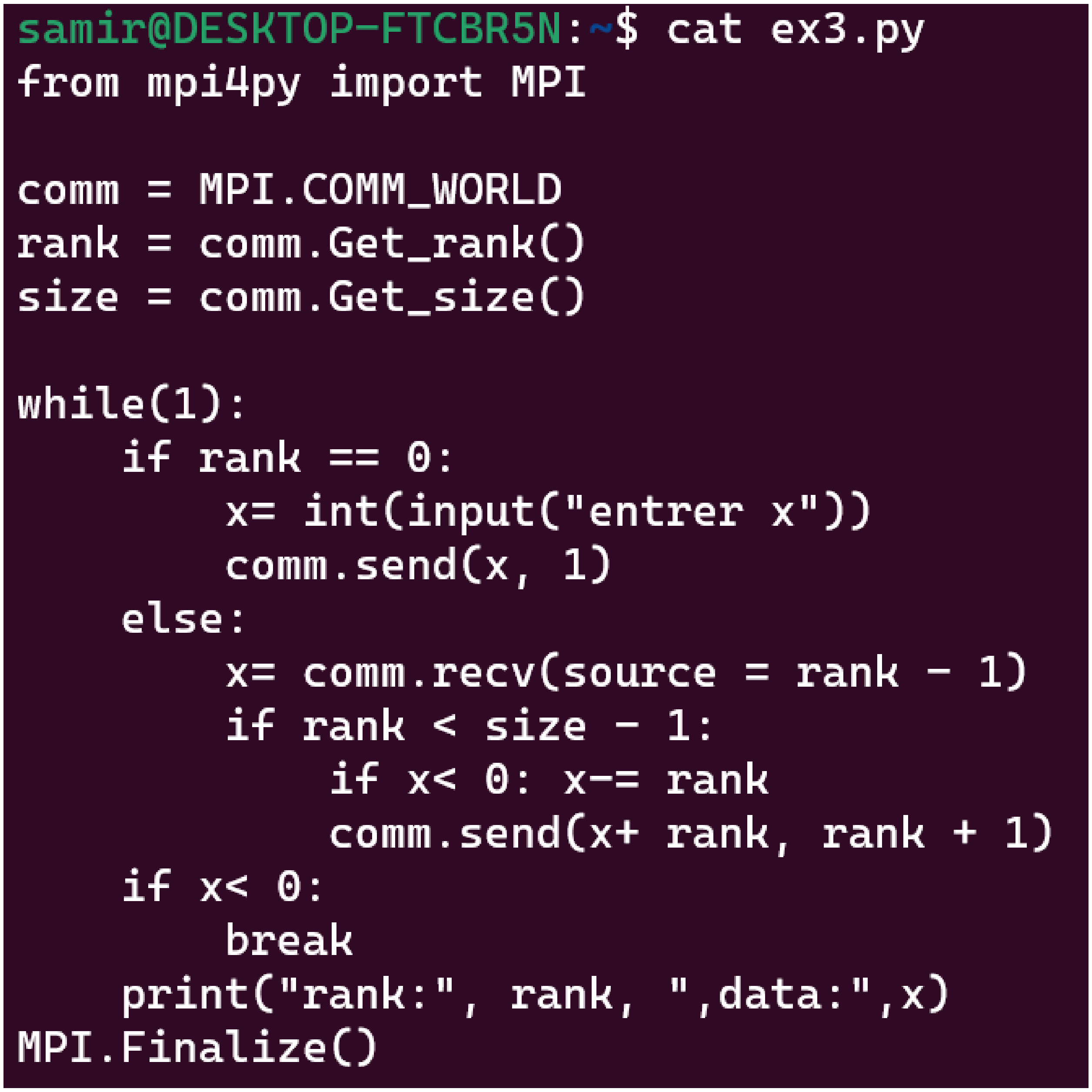
Exercice 2



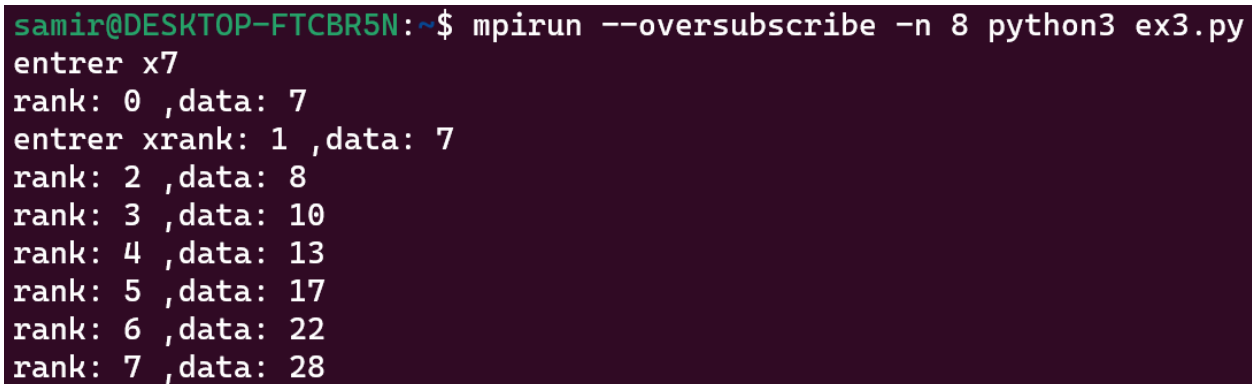
```
samir@DESKTOP-FTCBR5N:~$ cat ex2.py
from mpi4py import MPI

COMM =MPI.COMM_WORLD
RANK =COMM.Get_rank()
if RANK ==0:
    sendbuf=int(input("donner n"))
else:
    sendbuf =None
recvbuf =COMM.bcast(sendbuf, root=0)
print(RANK,recvbuf)
samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex2.py
donner n7
0 7
3 7
4 7
5 7
7 7
1 7
2 7
6 7
```

Exercice 3

Samir JABBAR



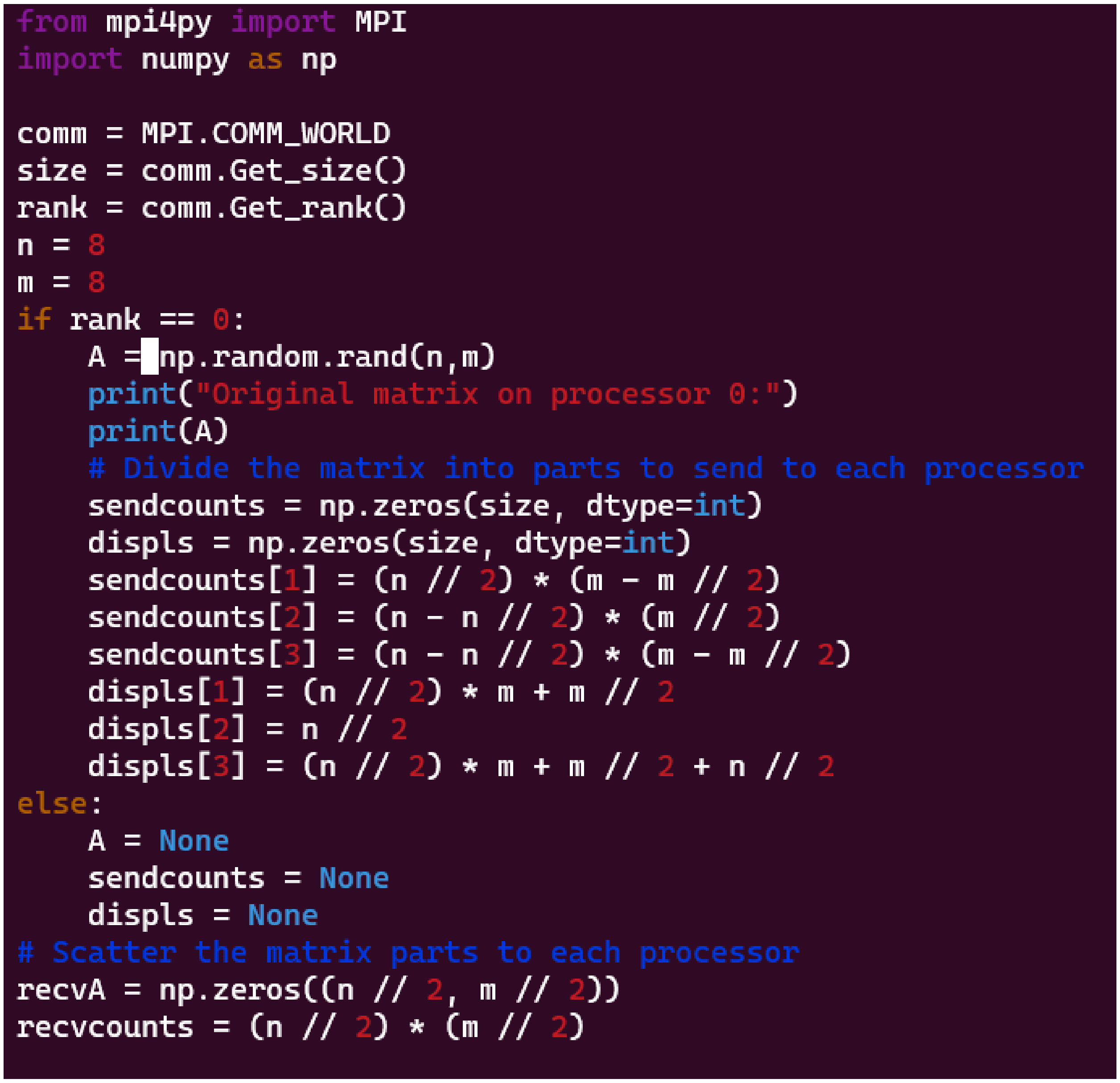
```
samir@DESKTOP-FTCBR5N:~$ cat ex3.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

while(1):
    if rank == 0:
        x= int(input("entrer x"))
        comm.send(x, 1)
    else:
        x= comm.recv(source = rank - 1)
        if rank < size - 1:
            if x< 0: x-= rank
            comm.send(x+ rank, rank + 1)
    if x< 0:
        break
    print("rank:", rank, ",data:",x)
MPI.Finalize()
```



```
samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex3.py
entrer x7
rank: 0 ,data: 7
entrer xrank: 1 ,data: 7
rank: 2 ,data: 8
rank: 3 ,data: 10
rank: 4 ,data: 13
rank: 5 ,data: 17
rank: 6 ,data: 22
rank: 7 ,data: 28
```

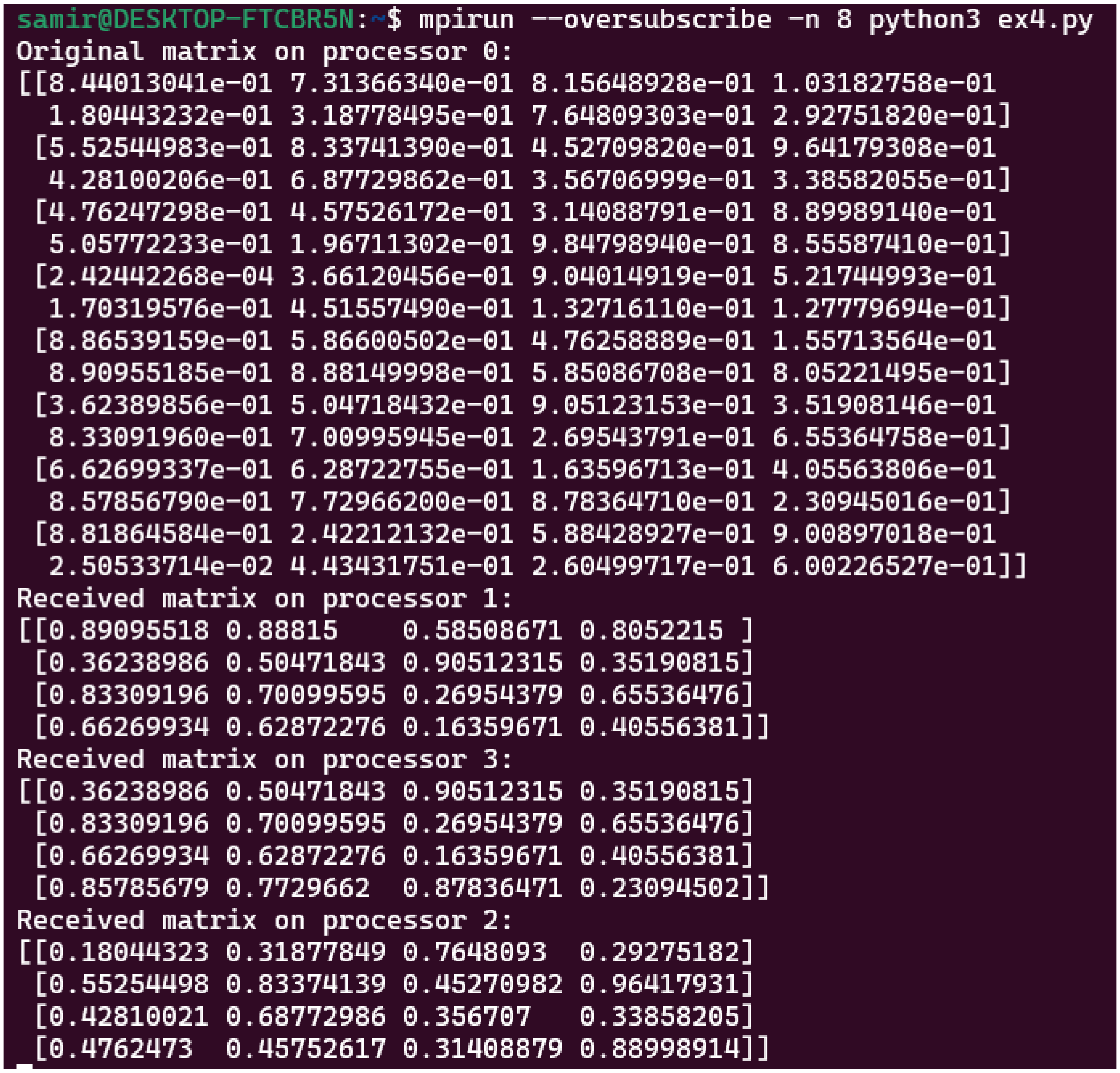
Samir JABBAR

Exercice 4



```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
n = 8
m = 8
if rank == 0:
    A = np.random.rand(n,m)
    print("Original matrix on processor 0:")
    print(A)
    # Divide the matrix into parts to send to each processor
    sendcounts = np.zeros(size, dtype=int)
    displs = np.zeros(size, dtype=int)
    sendcounts[1] = (n // 2) * (m - m // 2)
    sendcounts[2] = (n - n // 2) * (m // 2)
    sendcounts[3] = (n - n // 2) * (m - m // 2)
    displs[1] = (n // 2) * m + m // 2
    displs[2] = n // 2
    displs[3] = (n // 2) * m + m // 2 + n // 2
else:
    A = None
    sendcounts = None
    displs = None
# Scatter the matrix parts to each processor
recvA = np.zeros((n // 2, m // 2))
recvcounts = (n // 2) * (m // 2)
```

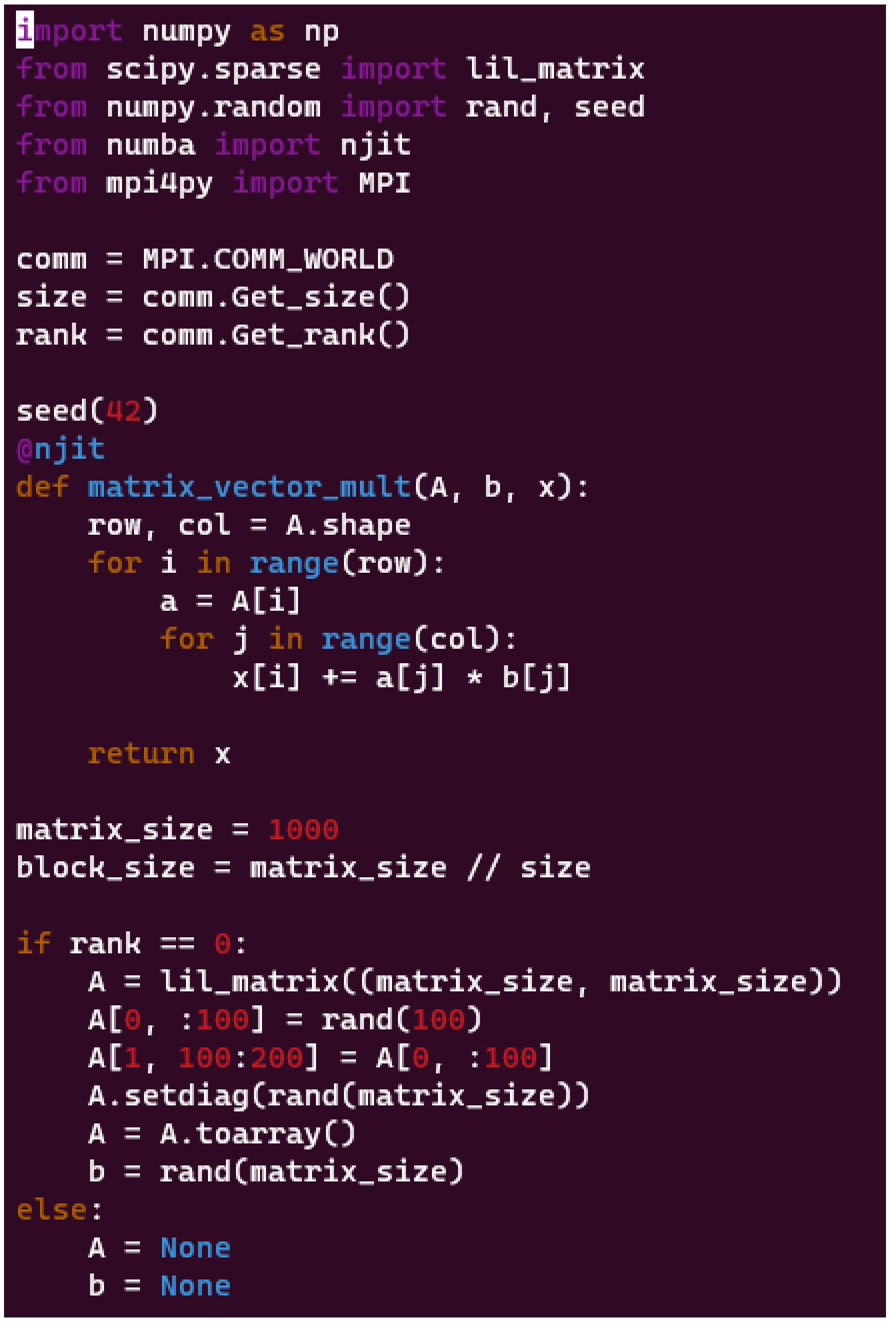
Samir JABBAR



```
samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex4.py
Original matrix on processor 0:
[[8.44013041e-01 7.31366340e-01 8.15648928e-01 1.03182758e-01
  1.80443232e-01 3.18778495e-01 7.64809303e-01 2.92751820e-01]
 [5.52544983e-01 8.33741390e-01 4.52709820e-01 9.64179308e-01
  4.28100206e-01 6.87729862e-01 3.56706999e-01 3.38582055e-01]
 [4.76247298e-01 4.57526172e-01 3.14088791e-01 8.89989140e-01
  5.05772233e-01 1.96711302e-01 9.84798940e-01 8.55587410e-01]
 [2.42442268e-04 3.66120456e-01 9.04014919e-01 5.21744993e-01
  1.70319576e-01 4.51557490e-01 1.32716110e-01 1.27779694e-01]
 [8.86539159e-01 5.86600502e-01 4.76258889e-01 1.55713564e-01
  8.90955185e-01 8.88149998e-01 5.85086708e-01 8.05221495e-01]
 [3.62389856e-01 5.04718432e-01 9.05123153e-01 3.51908146e-01
  8.33091960e-01 7.00995945e-01 2.69543791e-01 6.55364758e-01]
 [6.62699337e-01 6.28722755e-01 1.63596713e-01 4.05563806e-01
  8.57856790e-01 7.72966200e-01 8.78364710e-01 2.30945016e-01]
 [8.81864584e-01 2.42212132e-01 5.88428927e-01 9.00897018e-01
  2.50533714e-02 4.43431751e-01 2.60499717e-01 6.00226527e-01]]
Received matrix on processor 1:
[[0.89095518 0.88815    0.58508671 0.8052215 ]
 [0.36238986 0.50471843 0.90512315 0.35190815]
 [0.83309196 0.70099595 0.26954379 0.65536476]
 [0.66269934 0.62872276 0.16359671 0.40556381]]
Received matrix on processor 3:
[[0.36238986 0.50471843 0.90512315 0.35190815]
 [0.83309196 0.70099595 0.26954379 0.65536476]
 [0.66269934 0.62872276 0.16359671 0.40556381]
 [0.85785679 0.7729662  0.87836471 0.23094502]]
Received matrix on processor 2:
[[0.18044323 0.31877849 0.7648093  0.29275182]
 [0.55254498 0.83374139 0.45270982 0.96417931]
 [0.42810021 0.68772986 0.356707   0.33858205]
 [0.4762473  0.45752617 0.31408879 0.88998914]]
```

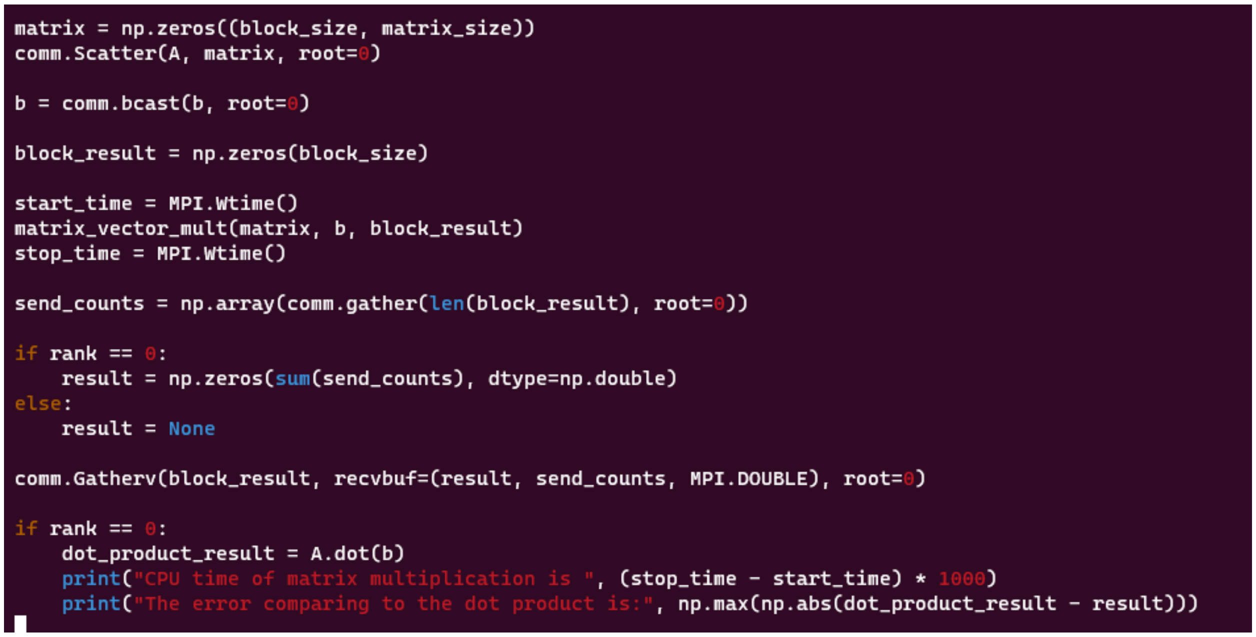
Exercice 5

Samir JABBAR



```python
import numpy as np
from scipy.sparse import lil_matrix
from numpy.random import rand, seed
from numba import njit
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

seed(42)
@njit
def matrix_vector_mult(A, b, x):
    row, col = A.shape
    for i in range(row):
        a = A[i]
        for j in range(col):
            x[i] += a[j] * b[j]

    return x

matrix_size = 1000
block_size = matrix_size // size

if rank == 0:
    A = lil_matrix((matrix_size, matrix_size))
    A[0, :100] = rand(100)
    A[1, 100:200] = A[0, :100]
    A.setdiag(rand(matrix_size))
    A = A.toarray()
    b = rand(matrix_size)
else:
    A = None
    b = None
```

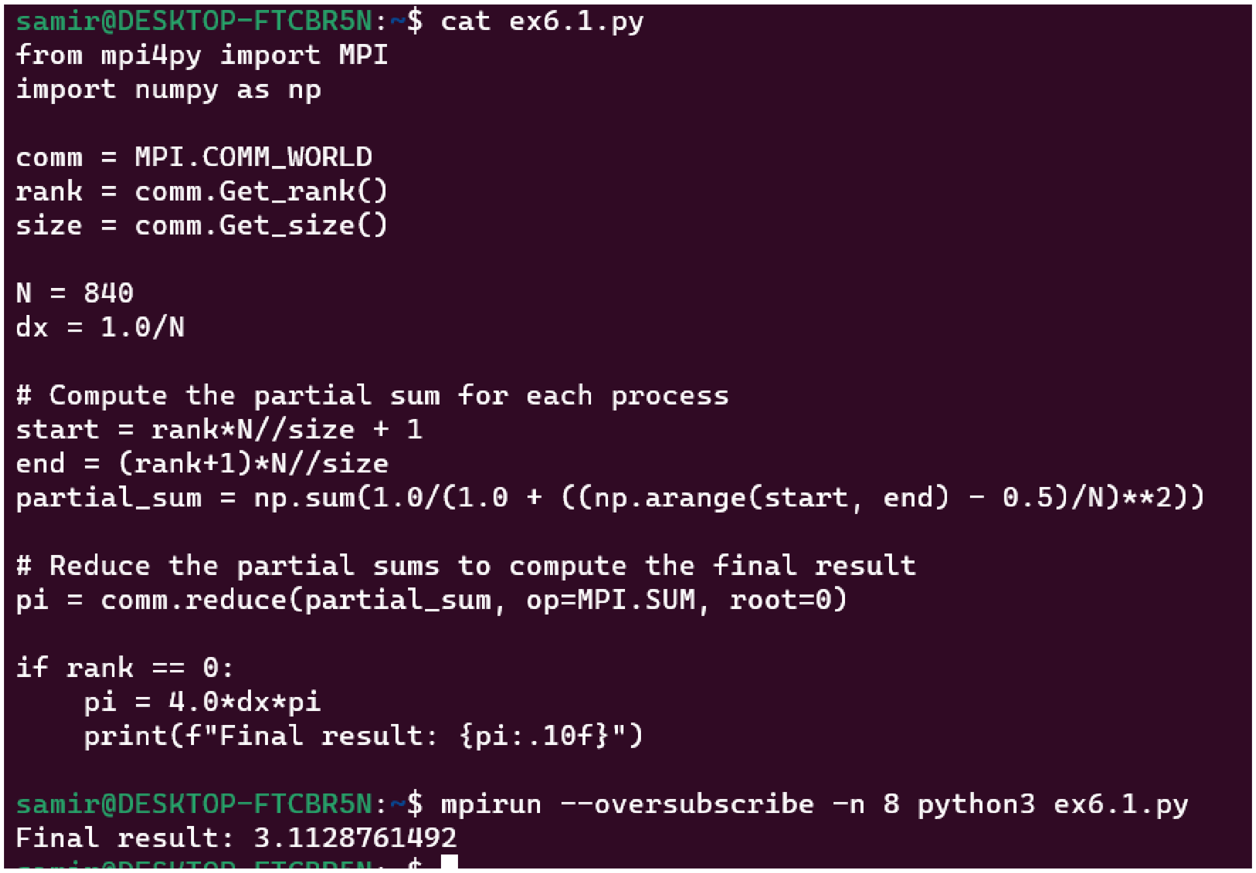
Samir JABBAR



```
matrix = np.zeros((block_size, matrix_size))
comm.Scatter(A, matrix, root=0)

b = comm.bcast(b, root=0)

block_result = np.zeros(block_size)

start_time = MPI.Wtime()
matrix_vector_mult(matrix, b, block_result)
stop_time = MPI.Wtime()

send_counts = np.array(comm.gather(len(block_result), root=0))

if rank == 0:
    result = np.zeros(sum(send_counts), dtype=np.double)
else:
    result = None

comm.Gatherv(block_result, recvbuf=(result, send_counts, MPI.DOUBLE), root=0)

if rank == 0:
    dot_product_result = A.dot(b)
    print("CPU time of matrix multiplication is ", (stop_time - start_time) * 1000)
    print("The error comparing to the dot product is:", np.max(np.abs(dot_product_result - result)))
```

Exercice 6

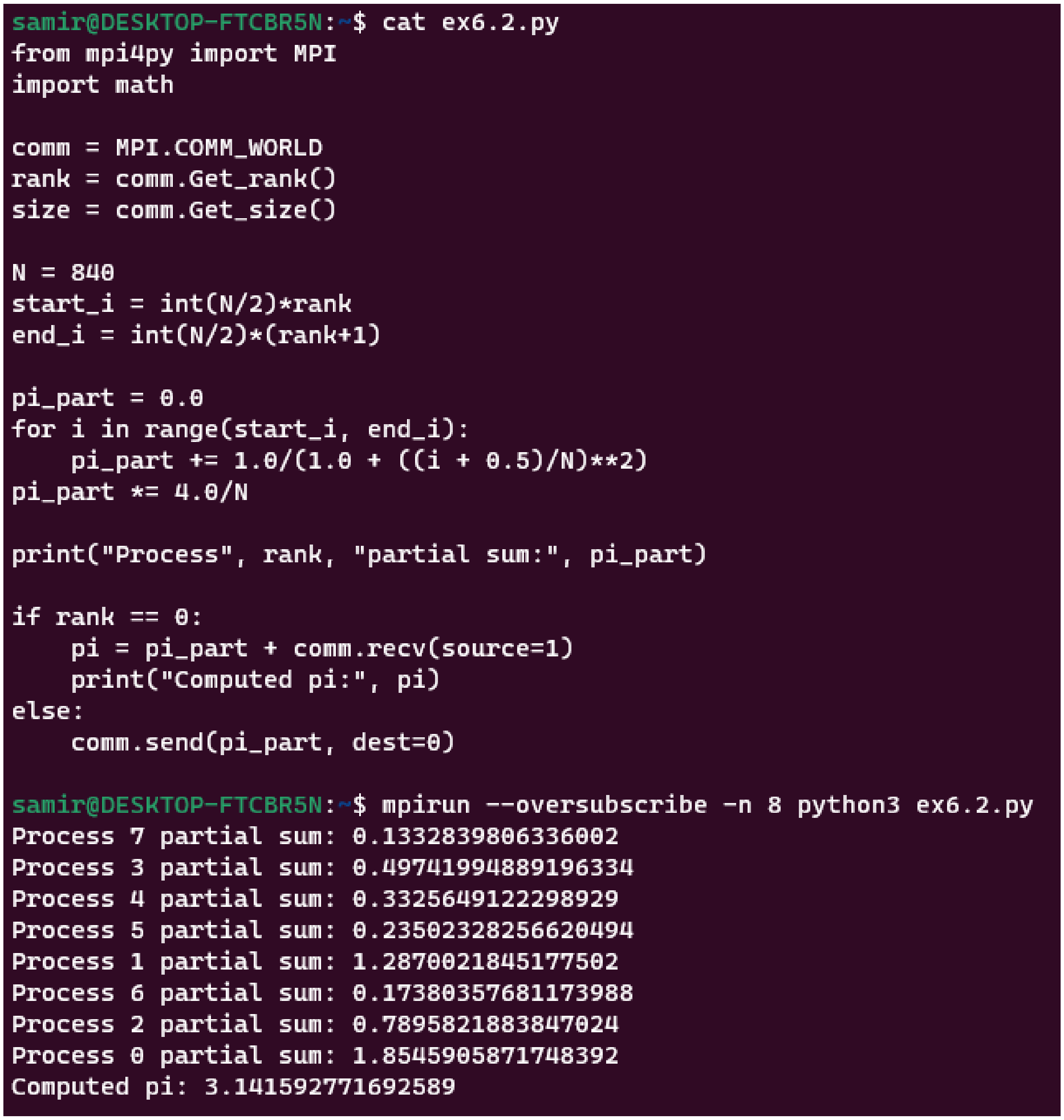
Q1



```
samir@DESKTOP-FTCBR5N:~$ cat ex6.1.py
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

N = 840
dx = 1.0/N

# Compute the partial sum for each process
start = rank*N//size + 1
end = (rank+1)*N//size
partial_sum = np.sum(1.0/(1.0 + ((np.arange(start, end) - 0.5)/N)**2))

# Reduce the partial sums to compute the final result
pi = comm.reduce(partial_sum, op=MPI.SUM, root=0)

if rank == 0:
    pi = 4.0*dx*pi
    print(f"Final result: {pi:.10f}")

samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex6.1.py
Final result: 3.1128761492
```

Samir JABBAR

Q2



```
samir@DESKTOP-FTCBR5N:~$ cat ex6.2.py
from mpi4py import MPI
import math

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

N = 840
start_i = int(N/2)*rank
end_i = int(N/2)*(rank+1)

pi_part = 0.0
for i in range(start_i, end_i):
    pi_part += 1.0/(1.0 + ((i + 0.5)/N)**2)
pi_part *= 4.0/N

print("Process", rank, "partial sum:", pi_part)

if rank == 0:
    pi = pi_part + comm.recv(source=1)
    print("Computed pi:", pi)
else:
    comm.send(pi_part, dest=0)

samir@DESKTOP-FTCBR5N:~$ mpirun --oversubscribe -n 8 python3 ex6.2.py
Process 7 partial sum: 0.1332839806336002
Process 3 partial sum: 0.49741994889196334
Process 4 partial sum: 0.3325649122298929
Process 5 partial sum: 0.23502328256620494
Process 1 partial sum: 1.2870021845177502
Process 6 partial sum: 0.17380357681173988
Process 2 partial sum: 0.7895821883847024
Process 0 partial sum: 1.8545905871748392
Computed pi: 3.141592771692589
```