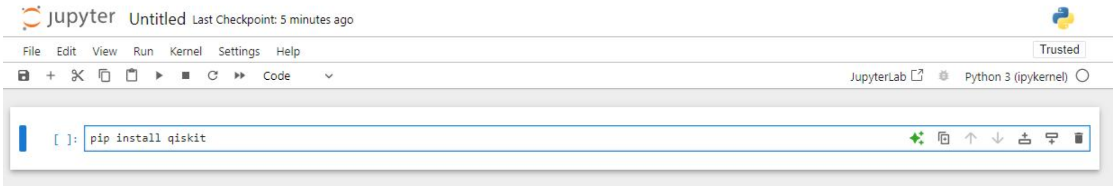
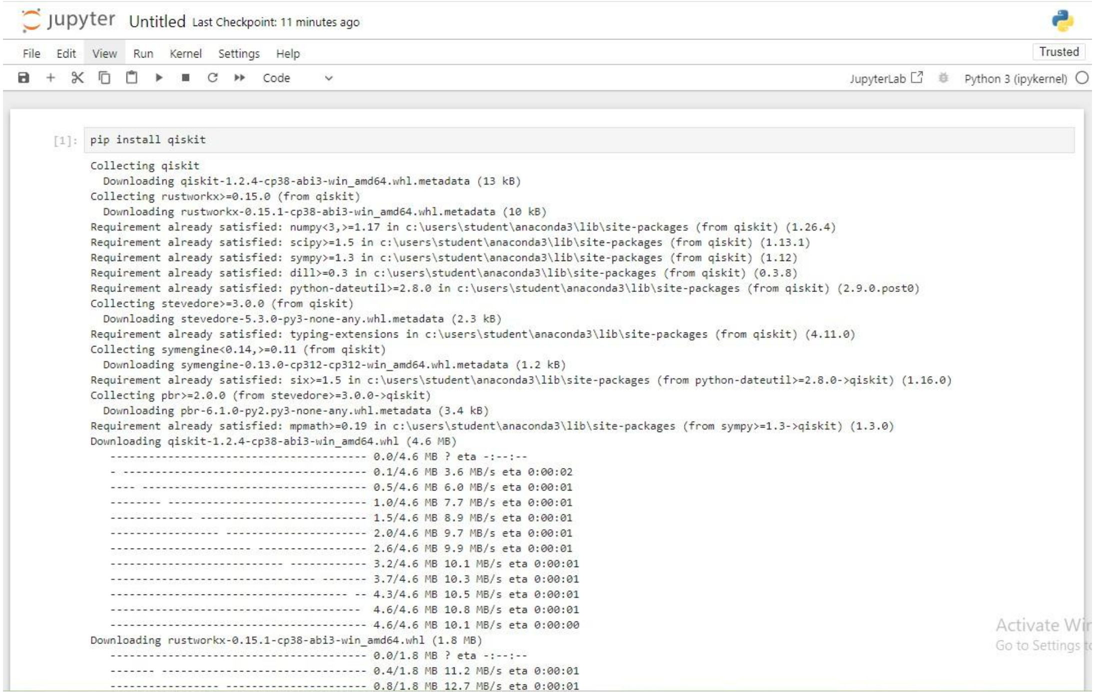
Name : - Ziya Pande
Roll no : - D17
SY-C COE

Experiment No 1: Installation of Qiskit





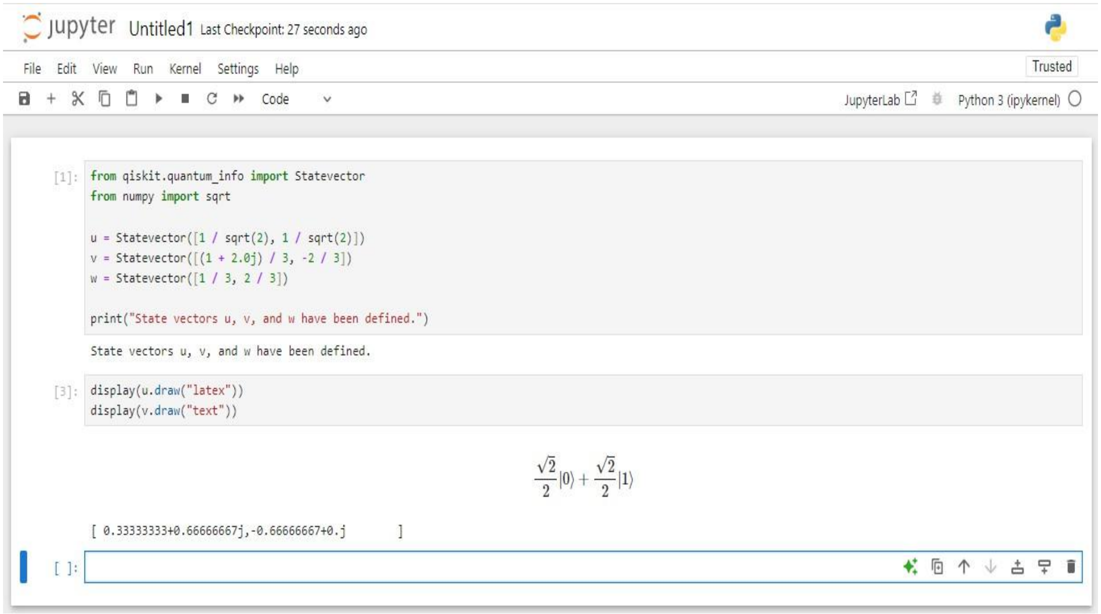
Name : - Ziya Pande
Roll no : - D17
SY-C COE

Experiment No 2: Linear Algebra, Vector Operation, Vector Multiplication, Tensor Product

Program:

```
from qiskit.quantum_info import Statevector
from numpy import sqrt

u = Statevector([1 / sqrt(2), 1 / sqrt(2)])
v = Statevector([(1 + 2.0j) / 3, -2 / 3])
w = Statevector([1 / 3, 2 / 3])

print("State vectors u, v, and w have been defined.")

display(u.draw("latex"))
display(v.draw("text"))
```

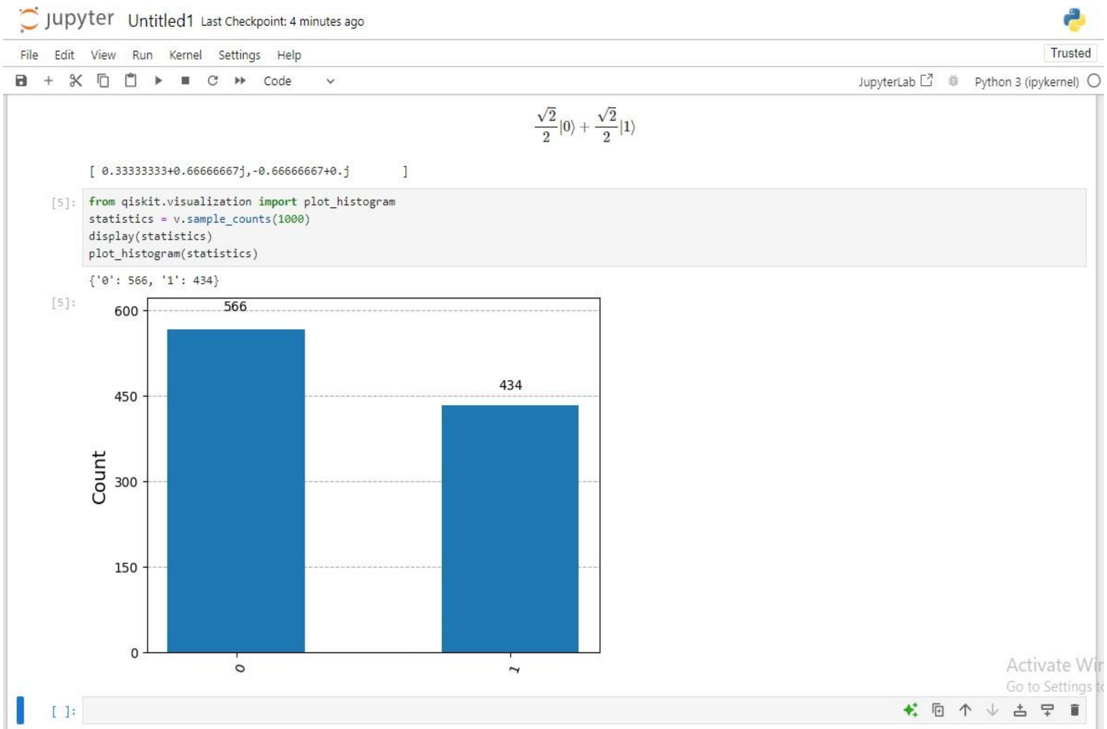
Output:



Program:

```
from qiskit.visualization import plot_histogram
statistics = v.sample_counts(1000)
display(statistics)
plot_histogram(statistics)
```

Output:



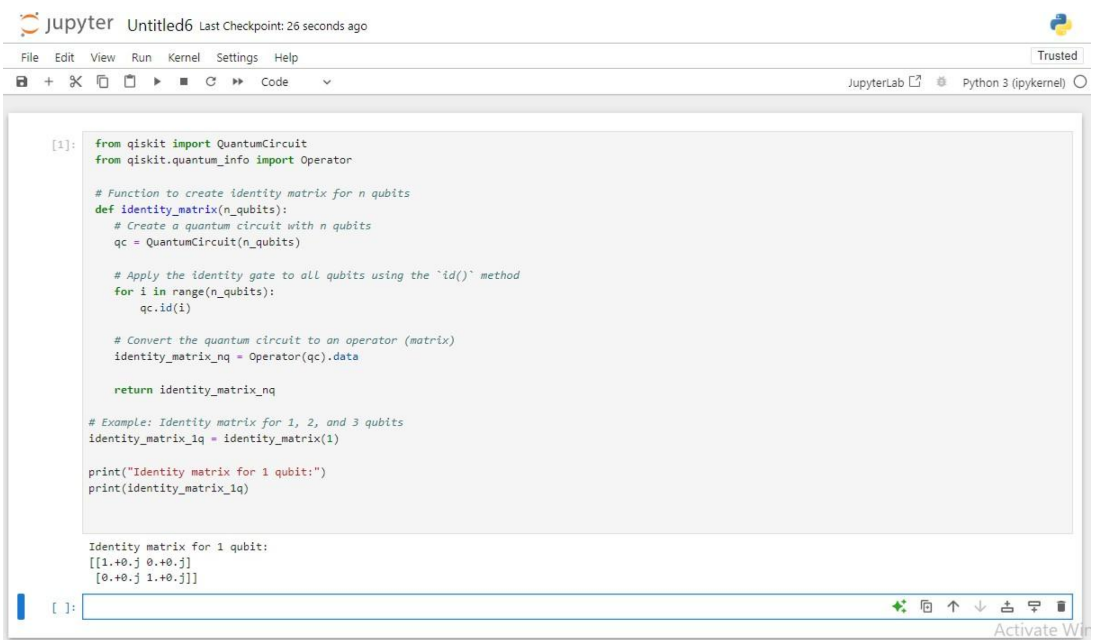
Name : - Ziya Pande
Roll no : - D17
SY-C COE

Experiment No 3: Implementation of Identity Matrix:1 Qubit, 2 Qubits, 3 Qubits

Program:

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Operator

# Function to create identity matrix for n qubits
def identity_matrix(n_qubits):
  # Create a quantum circuit with n qubits
  qc = QuantumCircuit(n_qubits)

  # Apply the identity gate to all qubits using the `id()` method
  for i in range(n_qubits):
     qc.id(i)

  # Convert the quantum circuit to an operator (matrix)
  identity_matrix_nq = Operator(qc).data

  return identity_matrix_nq

# Example: Identity matrix for 1, 2, and 3 qubits
identity_matrix_1q = identity_matrix(1)
identity_matrix_2q = identity_matrix(2)
identity_matrix_3q = identity_matrix(3)

print("Identity matrix for 1 qubit:")
print(identity_matrix_1q)

print("\nIdentity matrix for 2 qubits:")
print(identity_matrix_2q)

print("\nIdentity matrix for 3 qubits:")
print(identity_matrix_3q)
```

Output:



jupyter Untitled6 Last Checkpoint: 1 minute ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[3]:
```python
from qiskit import QuantumCircuit
from qiskit.quantum_info import Operator

# Function to create identity matrix for n qubits
def identity_matrix(n_qubits):
    # Create a quantum circuit with n qubits
    qc = QuantumCircuit(n_qubits)

    # Apply the identity gate to all qubits using the `id()` method
    for i in range(n_qubits):
        qc.id(i)

    # Convert the quantum circuit to an operator (matrix)
    identity_matrix_nq = Operator(qc).data

    return identity_matrix_nq

# Example: Identity matrix for 1, 2, and 3 qubits

identity_matrix_2q = identity_matrix(2)


print("\nIdentity matrix for 2 qubits:")
print(identity_matrix_2q)
```

```
Identity matrix for 2 qubits:
[[1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j]]
```


[ ]:

jupyter Untitled6 Last Checkpoint: 5 seconds ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[5]:
```python
from qiskit import QuantumCircuit
from qiskit.quantum_info import Operator

# Function to create identity matrix for n qubits
def identity_matrix(n_qubits):
    # Create a quantum circuit with n qubits
    qc = QuantumCircuit(n_qubits)

    # Apply the identity gate to all qubits using the `id()` method
    for i in range(n_qubits):
        qc.id(i)

    # Convert the quantum circuit to an operator (matrix)
    identity_matrix_nq = Operator(qc).data

    return identity_matrix_nq

# Example: Identity matrix for 1, 2, and 3 qubits
identity_matrix_3q = identity_matrix(3)

print("\nIdentity matrix for 3 qubits:")
print(identity_matrix_3q)
```

```
Identity matrix for 3 qubits:
[[1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j]]
```


[ ]:

Name : - Ziya Pande
Roll no : - D17
SY-C COE

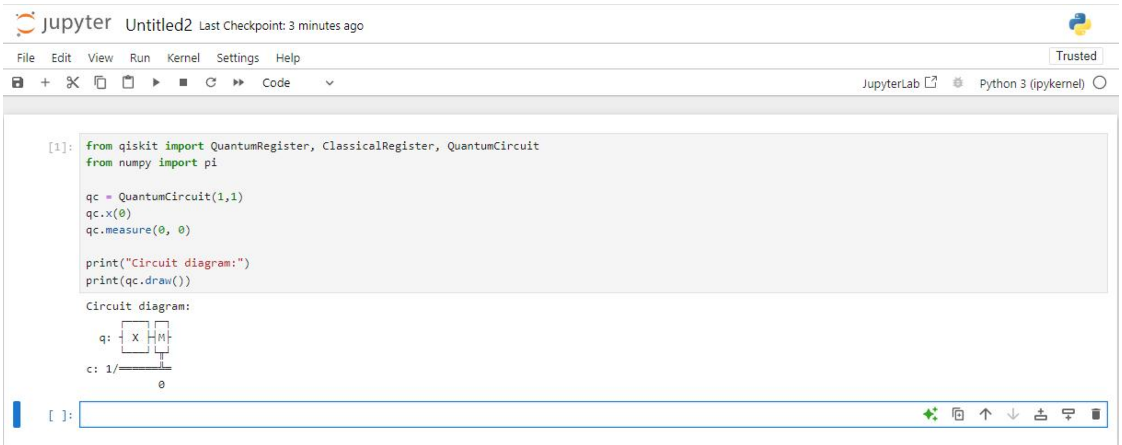
Experiment No 4: Implementation of Pauli Gates

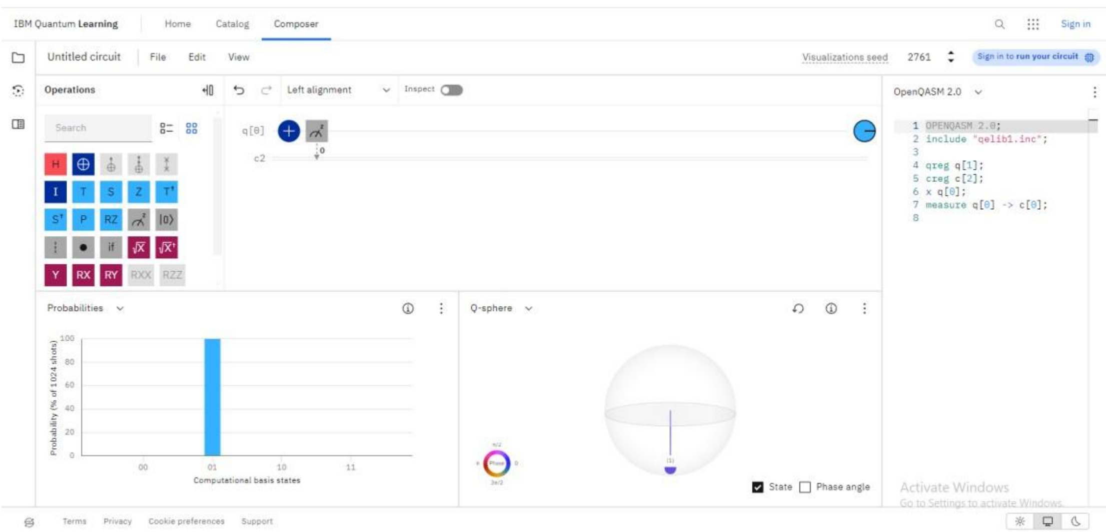
Program:

1. **Pauli-X Gate:**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(1,1)
qc.x(0)
qc.measure(0, 0)

print("Circuit diagram:")
print(qc.draw())
```

Output:

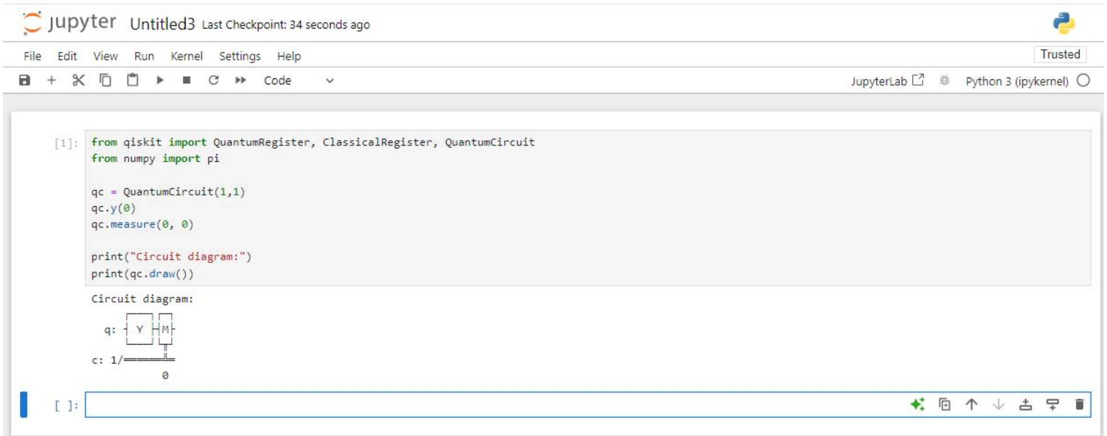


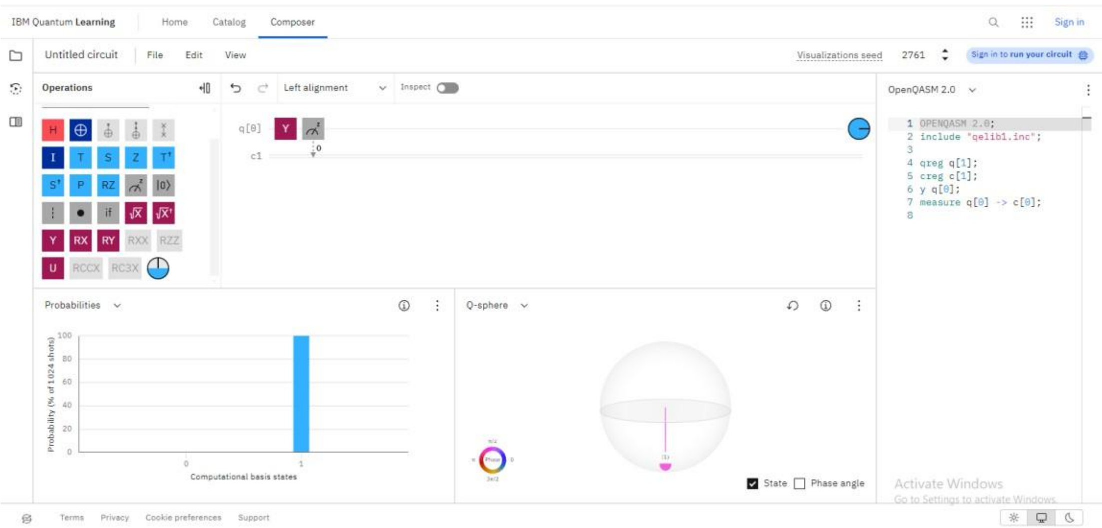


2. **Pauli-Y Gate:**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(1,1)
qc.y(0)
qc.measure(0, 0)

print("Circuit diagram:")
print(qc.draw())
```

Output:

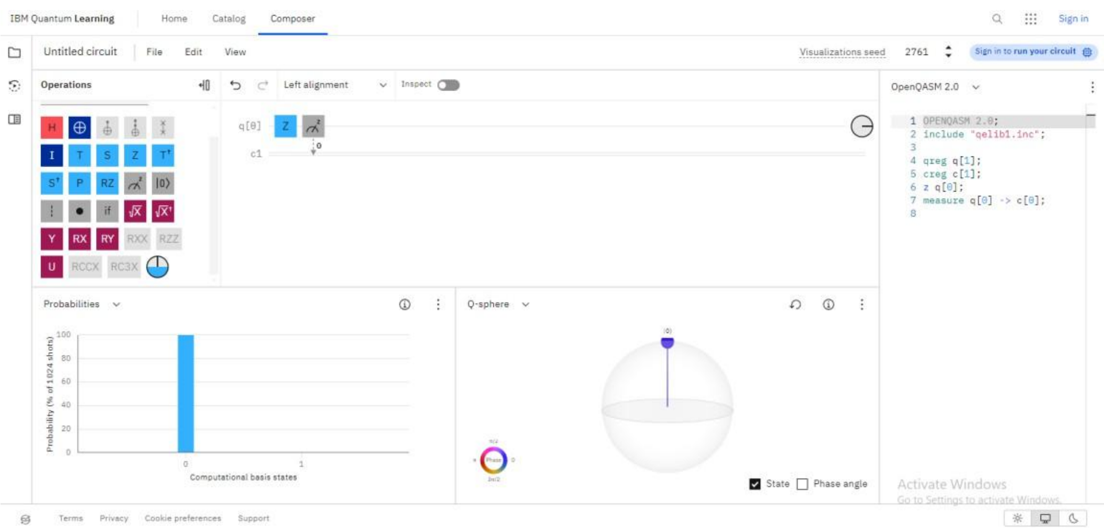




3. **Pauli-Z Gate:**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(1,1)
qc.y(0)
qc.measure(0, 0)

print("Circuit diagram:")
print(qc.draw())
```

Output:



Name : - Ziya Pande
Roll no : - D17
SY-C COE

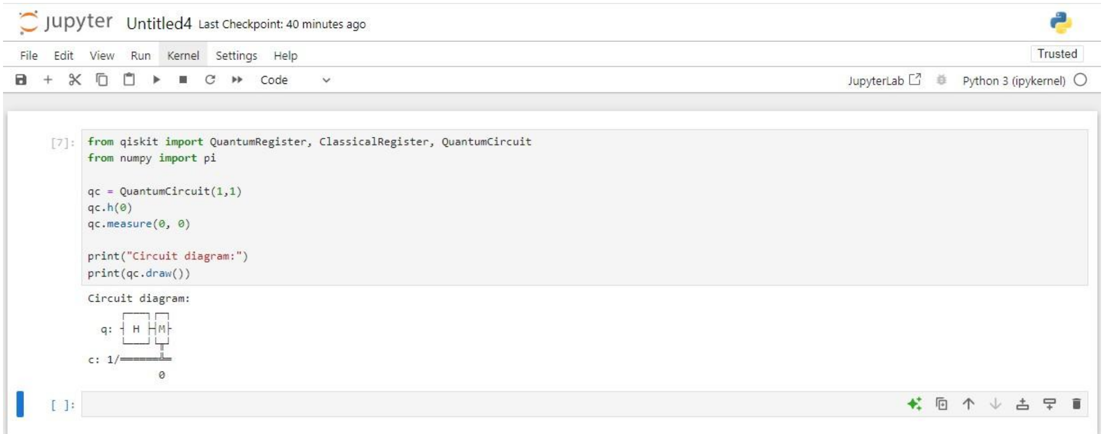
Experiment No 5: Implementation of Hadamard Gates

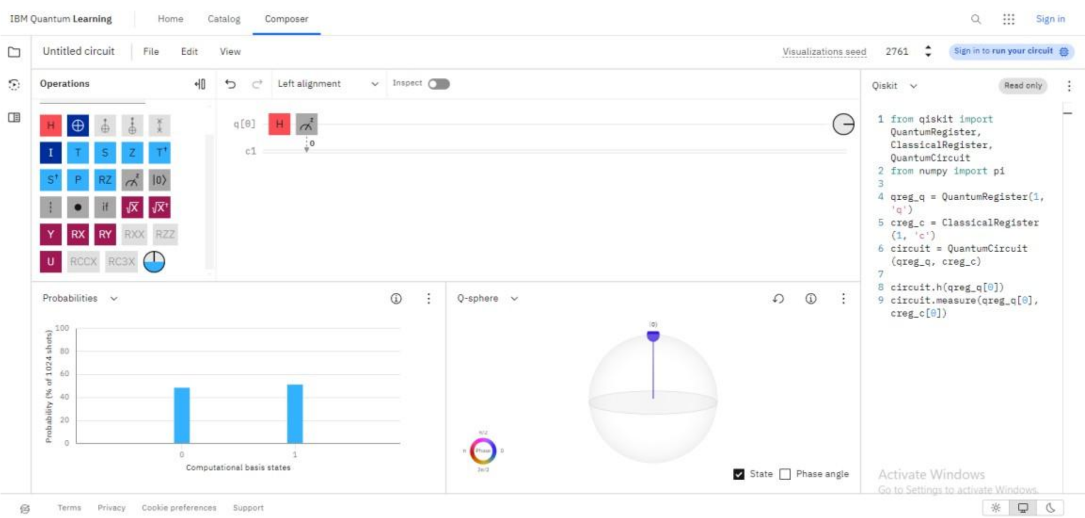
Program:

1. **Hadamard Gate (H):**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(1,1)
qc.h(0)
qc.measure(0, 0)

print("Circuit diagram:")
print(qc.draw())
```

Output:





Name : - Ziya Pande
Roll no : - D17
SY-C COE

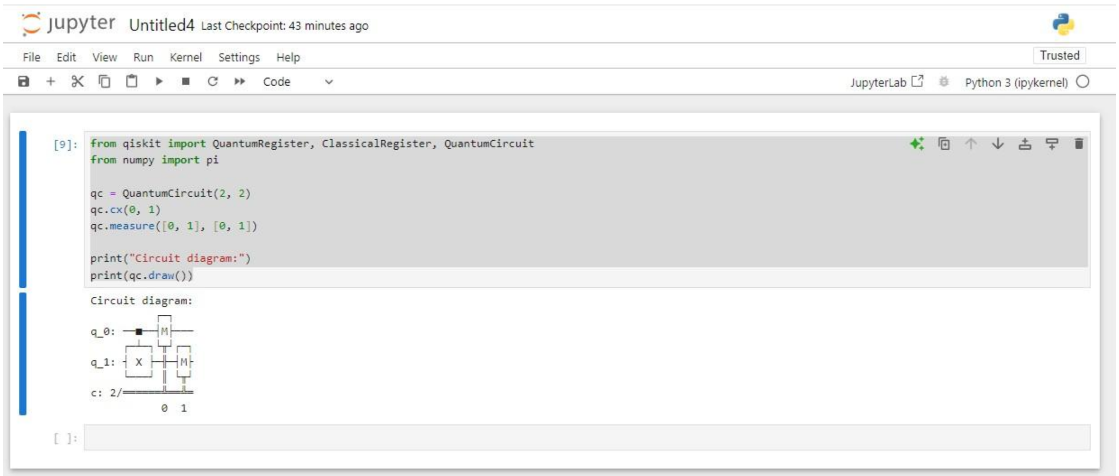
Experiment No 6: Implementation of 2 Qubit Gates

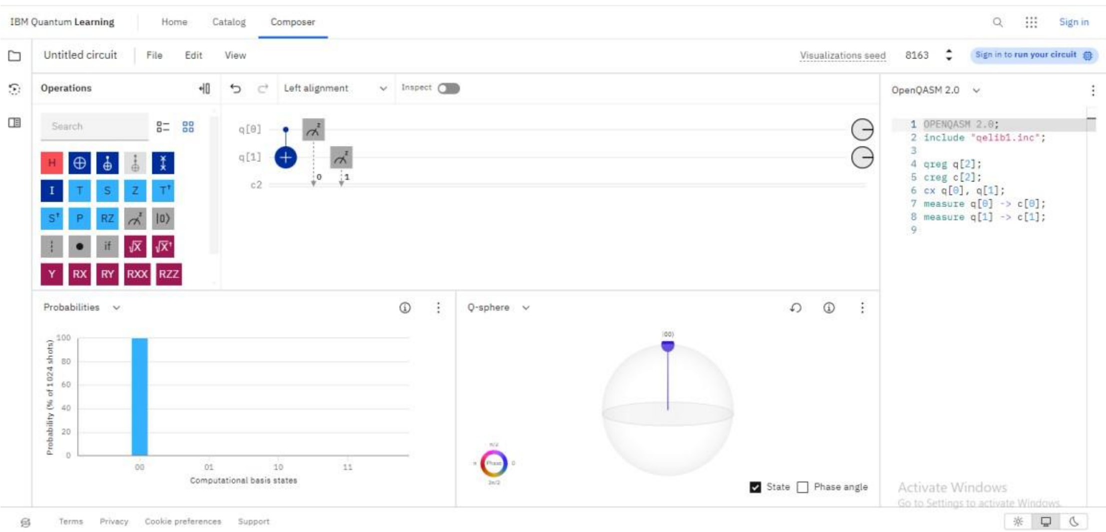
Program:

1. **CNOT Gate (CX):**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(2, 2)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])

print("Circuit diagram:")
print(qc.draw())
```

Output:



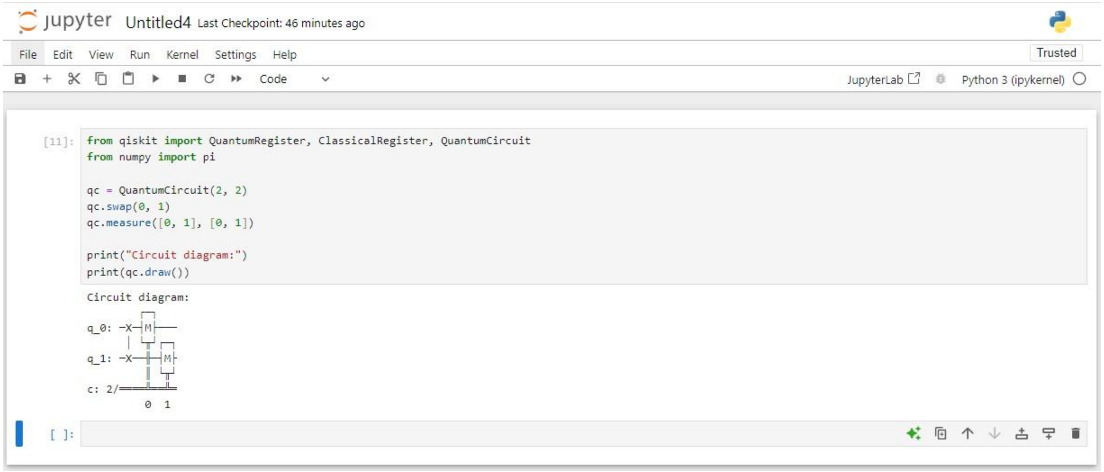


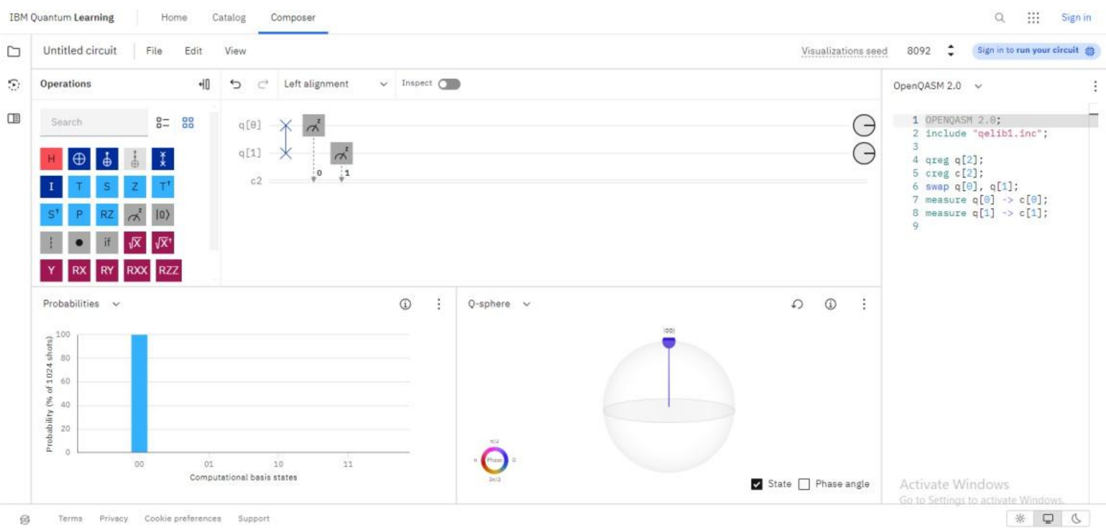
Program:

1. **SWAP Gate (SWAP):**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(2, 2)
qc.swap(0, 1)
qc.measure([0, 1], [0, 1])

print("Circuit diagram:")
print(qc.draw())
```

Output:





Name : - Ziya Pande
Roll no : - D17
SY-C COE

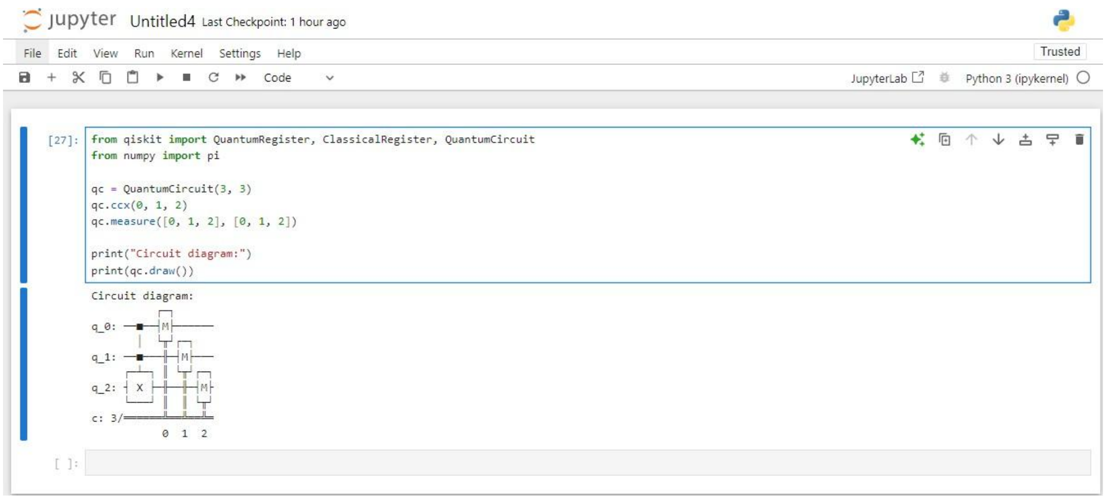
Experiment No 7: Implementation of 3 Qubit Gates

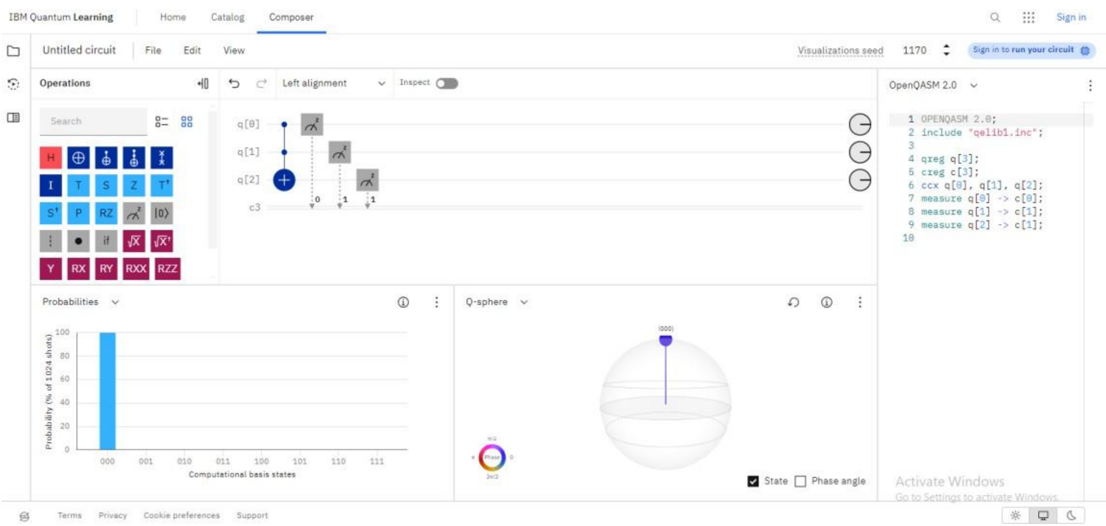
Program:

2. **TOFOLLI Gate (CCX):**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(3, 3)
qc.ccx(0, 1, 2)
qc.measure([0, 1, 2], [0, 1, 2])

print("Circuit diagram:")
print(qc.draw())
```

Output:





Name : - Ziya Pande
Roll no : - D17
SY-C COE

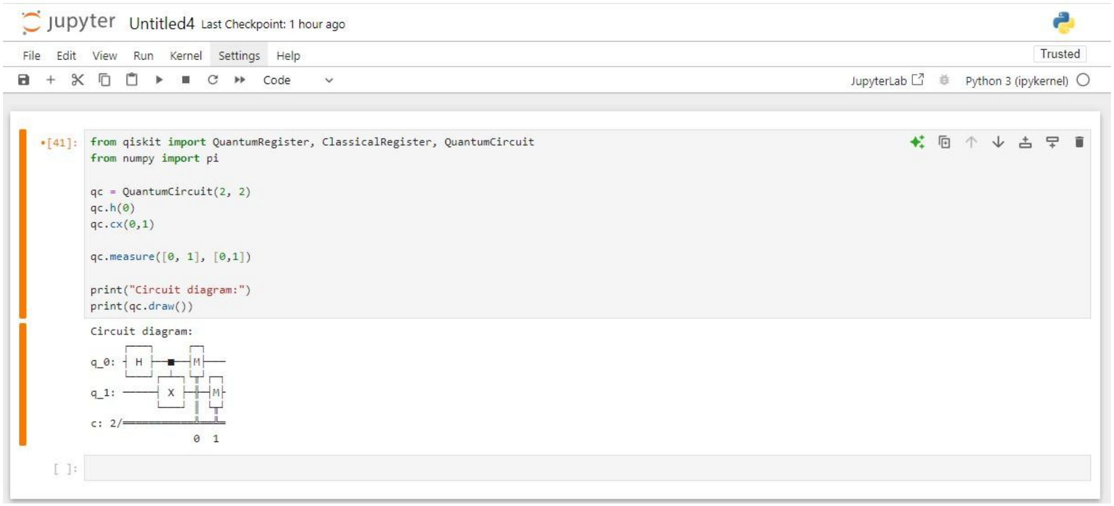
Experiment No 8: Implementation of Circuit Formation-1

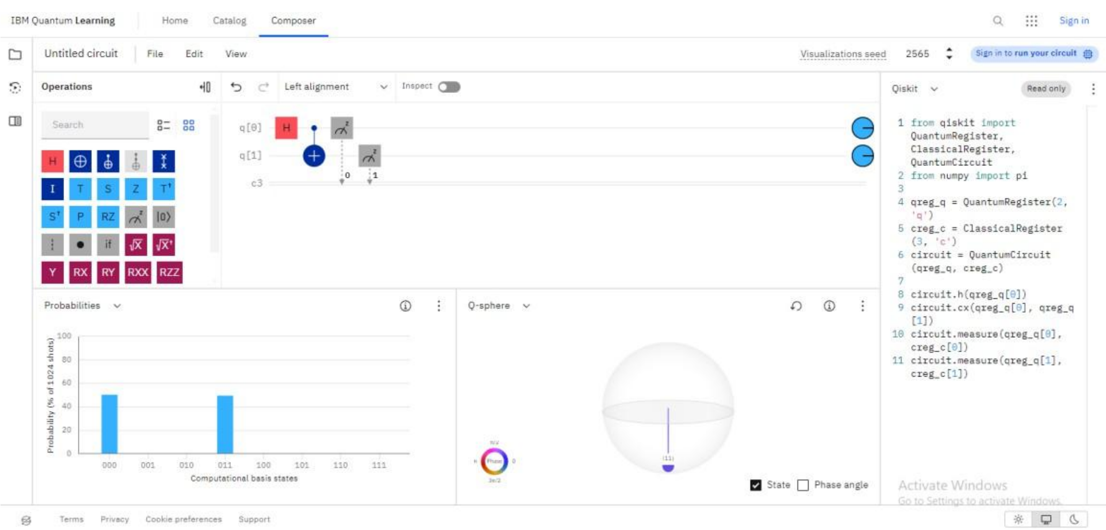
Program:

1. **Hadamard Gate on CNOT Gate (CX):**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0,1)

qc.measure([0, 1], [0,1])

print("Circuit diagram:")
print(qc.draw())
```

Output:

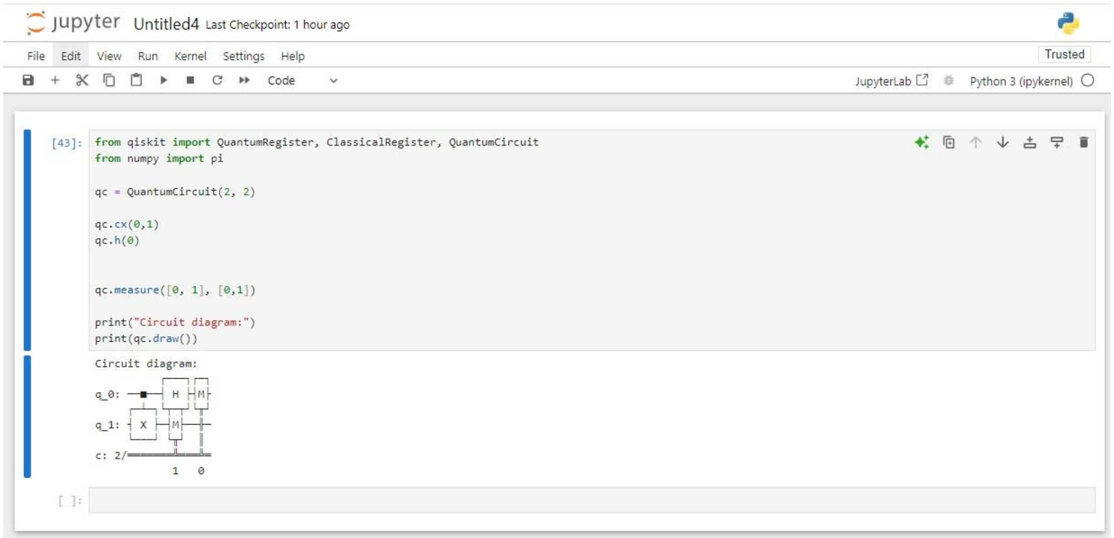


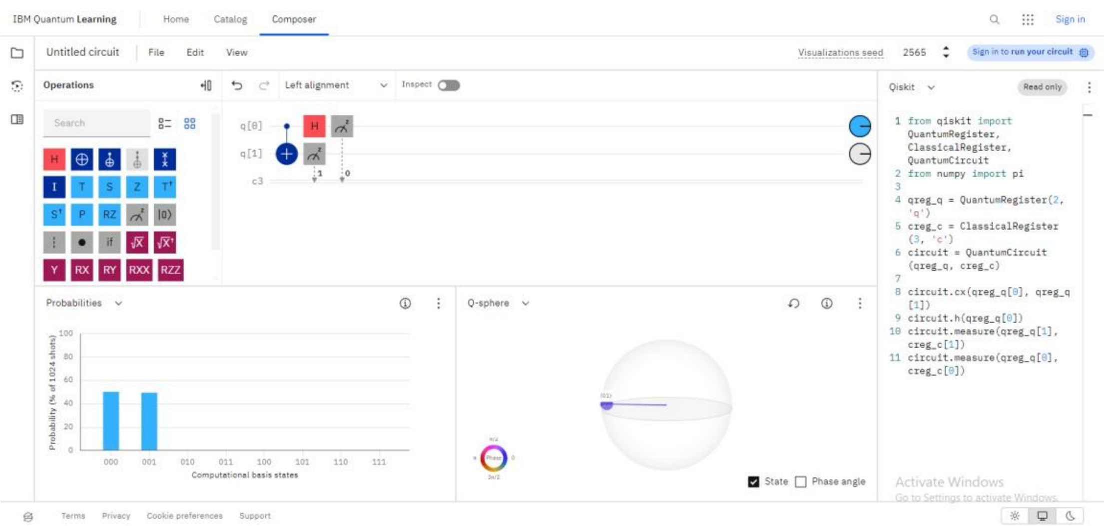


2. **CNOT Gate on Hadamard Gate (CX):**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(2, 2)

qc.cx(0,1)
qc.h(0)

qc.measure([0, 1], [0,1])

print("Circuit diagram:")
print(qc.draw())
```

Output:





Name : - Ziya Pande
Roll no : - D17
SY-C COE

Experiment No 9: Implementation of Circuit Formation-2

Program:

1. **2 Pauli-X gates on CCX Gate :**

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qc = QuantumCircuit(3, 3)

qc.x(0)
qc.x(1)
qc.ccx(0, 1, 2)

qc.measure([0, 1, 2], [0, 1, 2])

print("Circuit diagram:")
print(qc.draw())
```

Output: