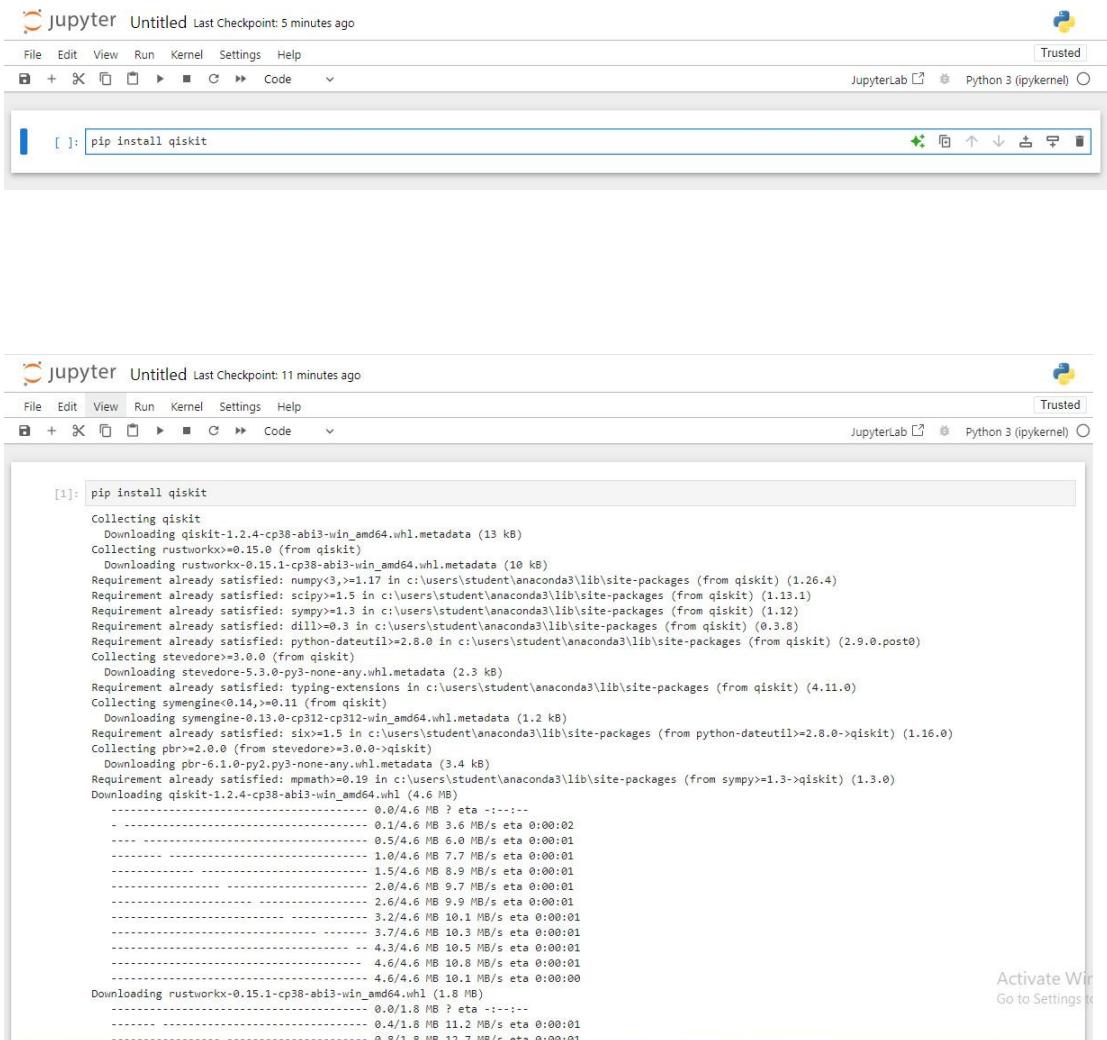


Name :- Ziya Pande  
Roll no :- D17  
SY-C COE

### Experiment No 1: Installation of Qiskit



The screenshot shows two consecutive screenshots of a Jupyter Notebook interface. Both screenshots have a title bar 'jupyter Untitled Last Checkpoint: [time ago]' and a menu bar 'File Edit View Run Kernel Settings Help'. A toolbar below the menu includes icons for file operations like new, open, save, and run, along with 'Code' and 'Cell' buttons. The status bar at the bottom right shows 'Trusted', 'JupyterLab', 'Python 3 (ipykernel)', and a kernel icon.

In the first screenshot, cell [1] contains the command 'pip install qiskit'. In the second screenshot, cell [1] shows the output of the pip install command, which details the download and installation of Qiskit and its dependencies. The output includes package names like qiskit, numpy, scipy, six, python-dateutil, pbr, and qiskit itself, along with their versions and download progress.

```
[1]: pip install qiskit
Collecting qiskit
  Downloading qiskit-1.2.4-cp38-abi3-win_amd64.whl.metadata (13 kB)
Collecting rustworkx==0.15.0 (from qiskit)
  Downloading rustworkx-0.15.1-cp38-abi3-win_amd64.whl.metadata (10 kB)
Requirement already satisfied: numpy<3,>=1.17 in c:\users\student\anaconda3\lib\site-packages (from qiskit) (1.26.4)
Requirement already satisfied: scipy>=1.5 in c:\users\student\anaconda3\lib\site-packages (from qiskit) (1.13.1)
Requirement already satisfied: sympy>=1.3 in c:\users\student\anaconda3\lib\site-packages (from qiskit) (1.12)
Requirement already satisfied: dlls=>0.3 in c:\users\student\anaconda3\lib\site-packages (from qiskit) (0.3.8)
Requirement already satisfied: python-dateutil>=2.8.0 in c:\users\student\anaconda3\lib\site-packages (from qiskit) (2.9.0.post0)
Collecting stevedore>=1.0.0 (from qiskit)
  Downloading stevedore-5.3.0-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: typing-extensions in c:\users\student\anaconda3\lib\site-packages (from qiskit) (4.11.0)
Collecting symengine<0.14,>=0.11 (from qiskit)
  Downloading symengine-0.13.0-cp312-cp312-win_amd64.whl.metadata (1.2 kB)
Requirement already satisfied: six>=1.5 in c:\users\student\anaconda3\lib\site-packages (from python-dateutil>=2.8.0->qiskit) (1.16.0)
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit)
  Downloading pbr-6.1.0-py2.py3-none-any.whl.metadata (3.4 kB)
Requirement already satisfied: mmaph>=0.19 in c:\users\student\anaconda3\lib\site-packages (from sympy>=1.3->qiskit) (1.3.0)
Downloading qiskit1.2.4-cp38-abi3-win_amd64.whl (4.6 MB)
----- 0.0/4.6 MB ? eta -:--:-
----- 0.1/4.6 MB 3.6 MB/s eta 0:00:02
----- 0.5/4.6 MB 6.0 MB/s eta 0:00:01
----- 1.0/4.6 MB 7.7 MB/s eta 0:00:01
----- 1.5/4.6 MB 8.9 MB/s eta 0:00:01
----- 2.0/4.6 MB 9.7 MB/s eta 0:00:01
----- 2.6/4.6 MB 9.9 MB/s eta 0:00:01
----- 3.2/4.6 MB 10.1 MB/s eta 0:00:01
----- 3.7/4.6 MB 10.3 MB/s eta 0:00:01
----- 4.3/4.6 MB 10.5 MB/s eta 0:00:01
----- 4.6/4.6 MB 10.8 MB/s eta 0:00:01
----- 4.6/4.6 MB 10.1 MB/s eta 0:00:00
----- 0.0/1.8 MB ? eta -:--:-
----- 0.4/1.8 MB 11.2 MB/s eta 0:00:01
----- 0.8/1.8 MB 12.7 MB/s eta 0:00:01
```

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:

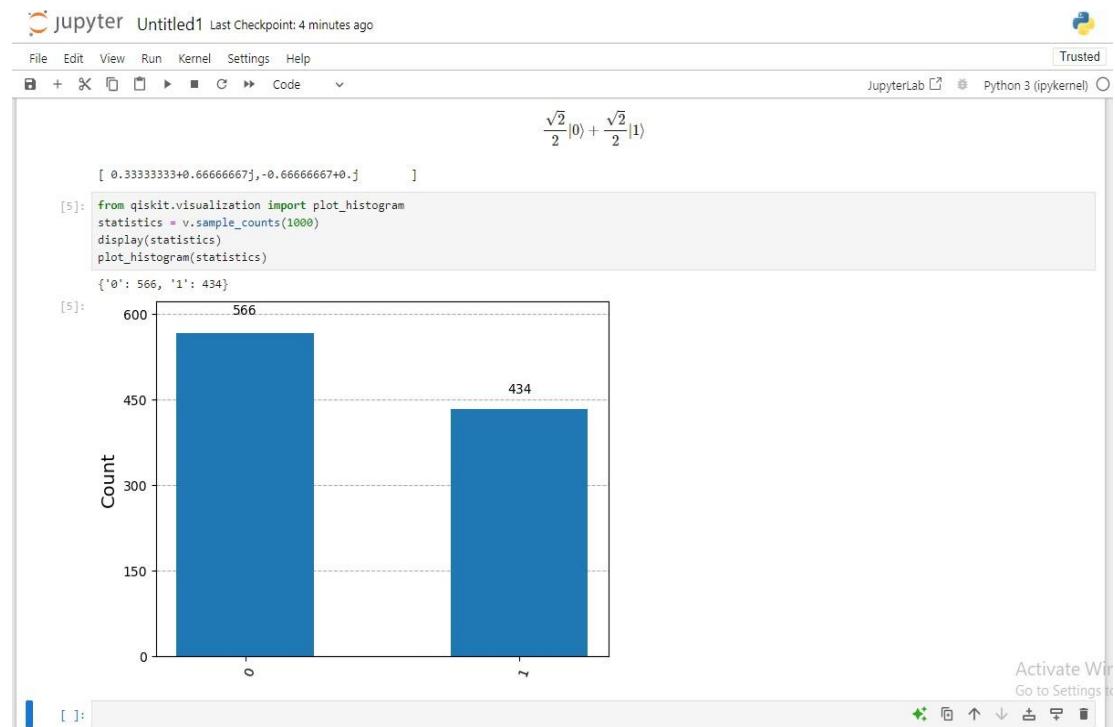
The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter Untitled1 Last Checkpoint: 27 seconds ago, Trusted, Python 3 (ipykernel)
- Toolbar:** File, Edit, View, Run, Kernel, Settings, Help
- Code Cell 1:** [1]:  
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.")  
  
State vectors u, v, and w have been defined.
- Code Cell 3:** [3]:  
display(u.draw("latex"))  
display(v.draw("text"))  
  
$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$
  
[ 0.33333333+0.66666667j, -0.66666667+0.j ]
- Bottom Bar:** [ ], navigation icons (back, forward, search, etc.)

**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

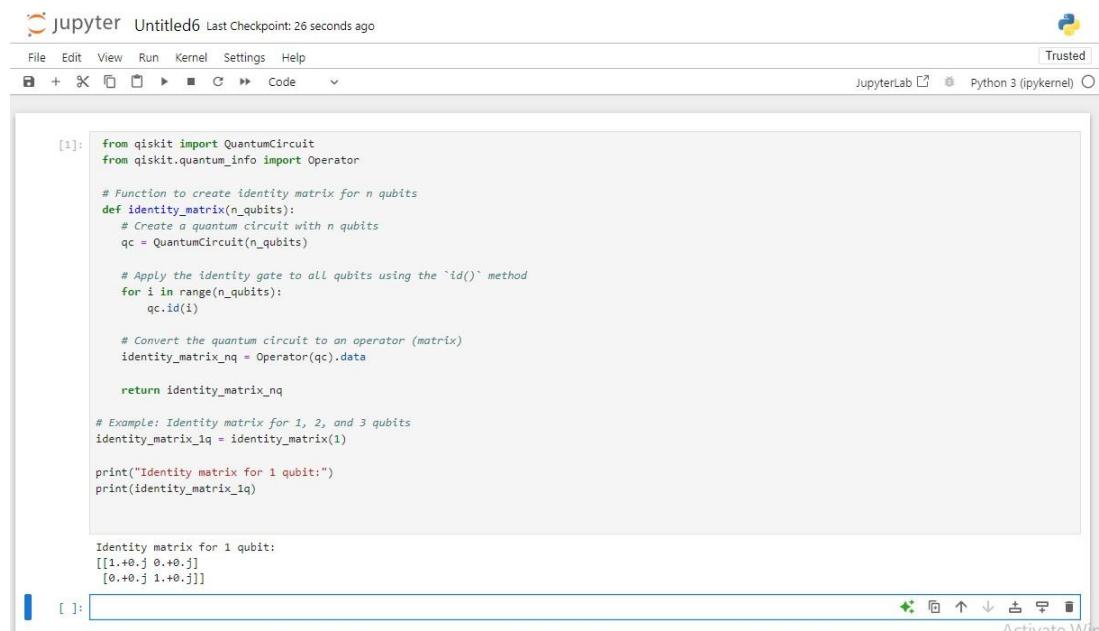
# Example: Identity matrix for 1, 2, and 3 qubits
identity_matrix_1q = identity_matrix(1)
identity_matrix_2q = identity_matrix(2)
identity_matrix_3q = identity_matrix(3)

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:



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter Untitled6 Last Checkpoint: 26 seconds ago
- Toolbar:** File Edit View Run Kernel Settings Help Trusted JupyterLab Python 3 (ipykernel)
- Code Cell:** [1]:

```
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)

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 Cell:** [1]:

```
Identity matrix for 1 qubit:
[[1.+0.j 0.+0.j]
 [0.+0.j 1.+0.j]]
```
- Bottom Bar:** Activate Win

jupyter Untitled6 Last Checkpoint: 1 minute ago

File Edit View Run Kernel Settings Help Trusted

[ ]:

```
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

JupyterLab Python 3 (ipykernel) C

```
[5]: 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:

$$[[1.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j]$$
$$[0.0+0.j 1.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j]$$
$$[0.0+0.j 0.0+0.j 1.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j]$$
$$[0.0+0.j 0.0+0.j 0.0+0.j 1.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j]$$
$$[0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 1.0+0.j 0.0+0.j 0.0+0.j]$$
$$[0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 1.0+0.j 0.0+0.j]$$
$$[0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 0.0+0.j 1.0+0.j]]$$

Activate W  
Go to Settings

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())
```

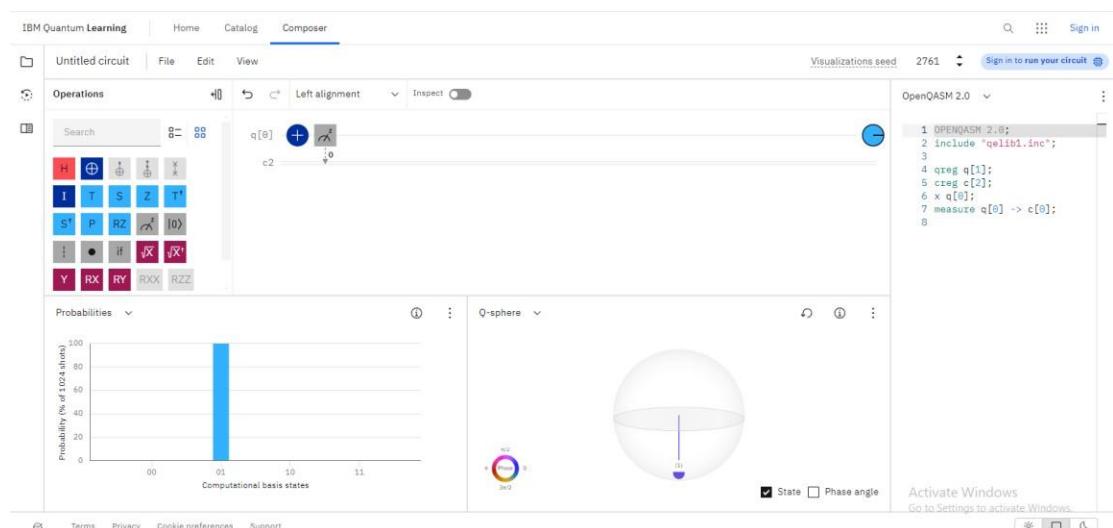
##### Output:

The screenshot shows a Jupyter Notebook cell with the following content:

```
[1]: 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())
```

Circuit diagram:

q: ┌── X ──┐  
 └─────────┘  
c: 1/ ────────── 0



## 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:

The screenshot shows a Jupyter Notebook cell with the following content:

```
[1]: 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())
```

Below the code, the output shows the circuit diagram:

Circuit diagram:

```
q: ┌───┐  
    ┌ Y ─┐  
    └───┘  
c: 1/─  
      0
```

The screenshot shows the IBM Quantum Learning interface with the following components:

- Operations Panel:** A grid of quantum gate icons including H, T, S, Z, T†, S†, RZ, RX, RY, RXX, RYY, RZZ, U, RCX, and RC3X.
- Quantum Circuit Editor:** A workspace where a circuit can be built by dragging gates onto a timeline.
- Probability Plot:** A bar chart titled "Probabilities (% of 1024 shots)" showing the distribution of computational basis states. The x-axis is "Computational basis states" (0, 1) and the y-axis is "Probability" (0, 100). The bar for state 1 reaches 100%.
- Q-sphere:** A 3D visualization of a Bloch sphere showing the state vector's phase and magnitude.
- OpenQASM 2.0 Editor:** A text area containing the generated OpenQASM 2.0 code.

```
1 OPENQASM 2.0;  
2 include "qelib1.inc";  
3  
4 qreg q[1];  
5 creg c[1];  
6 y q[0];  
7 measure q[0] -> c[0];  
8
```

### 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:

The screenshot shows a Jupyter Notebook cell with the following content:

```
[5]: 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())
```

Below the code, the output shows the circuit diagram:

Circuit diagram:

```
q: ┌───┐  
    ┌ Z ─┐  
    ┌ M ─┐  
    ┌───┐  
c: 1/ └───┘  
      0
```

The screenshot shows the IBM Quantum Learning Composer interface with the following components:

- Operations Panel:** A grid of icons representing various quantum gates and operations.
- Quantum Circuit Editor:** A workspace where a circuit is being built. The circuit consists of one qubit register (q[0]) and one classical bit register (c[0]). A Z gate is applied to q[0], followed by a measurement operation (M) that stores the result in c[0].
- OpenQASM 2.0 Editor:** A text area containing the OpenQASM 2.0 code for the circuit:

```
OPENQASM 2.0;  
include "qelib1.inc";  
qreg q[1];  
creg c[1];  
z q[0];  
measure q[0] -> c[0];
```
- Probability Plot:** A bar chart showing the probability distribution of computational basis states. The x-axis is labeled "Computational basis states" and has two ticks: 0 and 1. The y-axis is labeled "Probability (%) of showing" and ranges from 0 to 100. The bar for state 0 reaches 100%.
- Q-sphere:** A 3D visualization of the Bloch sphere showing the state vector |0>. The vector points along the positive z-axis.
- Activation Message:** A message at the bottom right corner says "Activate Windows Go to Settings to activate Windows".

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:")
```

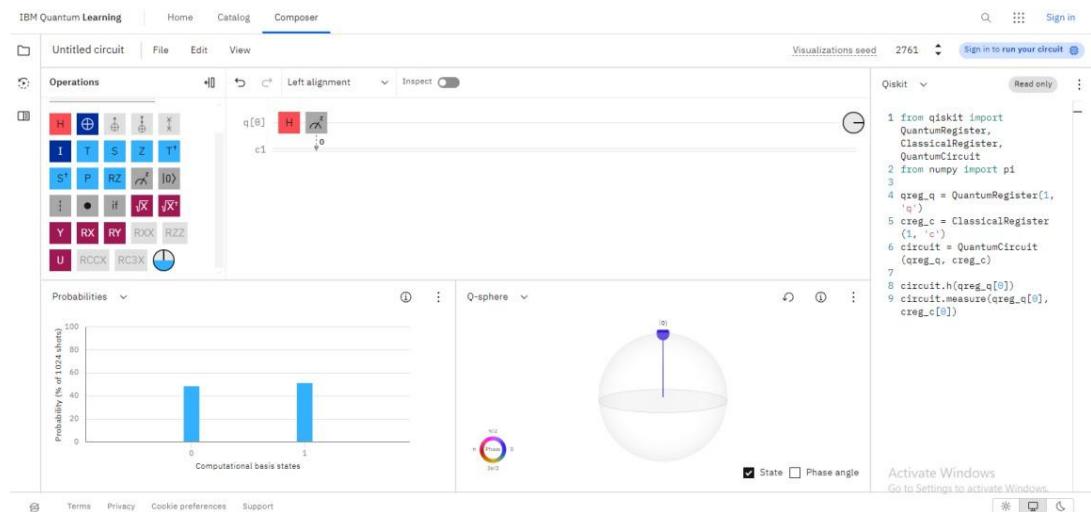
```
print(qc.draw())
```

#### Output:

```
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())
```

Circuit diagram:

```
q: ┌───┐  
   ┌ H ─┐  
   └───┘  
c: 1/─  
    0
```



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:

```
[9]: 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())

Circuit diagram:
q_0: ──■─────────
                  ┌─┐
q_1: ──X─■─────────
                  ┌─┐
c: 2/─■─────────
                0 1
```

The screenshot shows the IBM Quantum Learning interface with the following components:

- Circuit View:** Displays the quantum circuit with two qubits and one classical register. It shows a CNOT gate from qubit 0 to qubit 1, followed by a measurement of both qubits.
- QASM View:** Shows the OpenQASM 2.0 code generated by the circuit:

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
cx q[0], q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```
- Probability Plot:** A bar chart showing the probability distribution of the four computational basis states (00, 01, 10, 11). The state 00 has a probability of approximately 95%, while others are near zero.
- State Visualization:** A 3D sphere representing the state vector. The vector is primarily along the positive z-axis, indicating a high probability of the 00 state.

## Program:

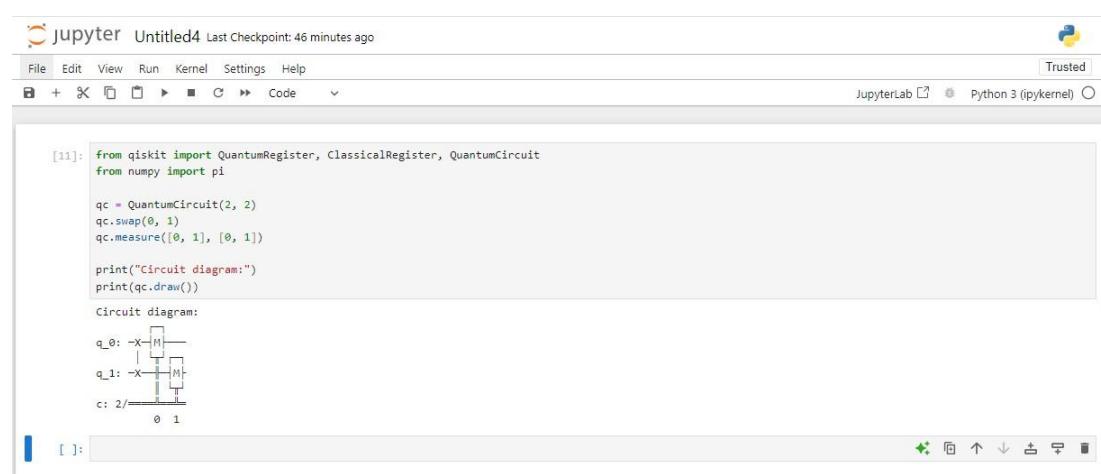
### 1. SWAP Gate (SWAP):

```
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:

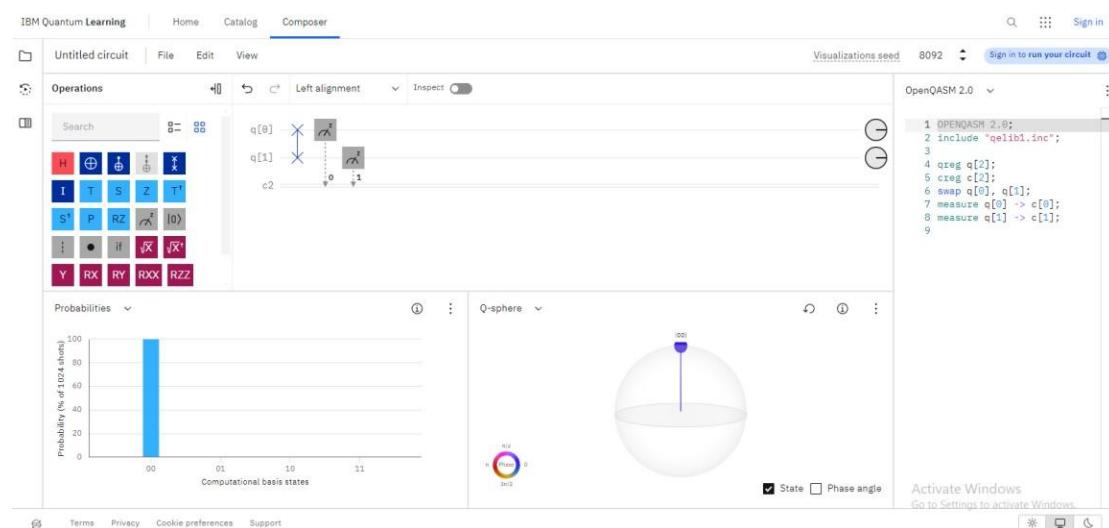


```
[11]: from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

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

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

Circuit diagram:
q_0: ┌───┐
      └───┘
q_1: ┌───┐
      └───┘
c: 2/─
      0 1
```



IBM Quantum Learning Home Catalog Composer

Operations Left alignment Inspect

Search

OpenQASM 2.0

```
1 OPENQASM 2.0;
2 include "qelib1.inc";
3
4 qreg q[2];
5 creg c[2];
6 swap q[0], q[1];
7 measure q[0] > c[0];
8 measure q[1] > c[1];
```

Visualizations seed 8092 Sign in to run your circuit

Probabilities Computational basis states

Computational basis states	Probability (%)
00	99
01	0
10	0
11	0

Q-sphere State Phase angle

Activate Windows Go to Settings to activate Windows

Name :- Ziya Pande

Roll no :- D17

SY-C COE

### Experiment No 7: Implementation of 3 Qubit Gates

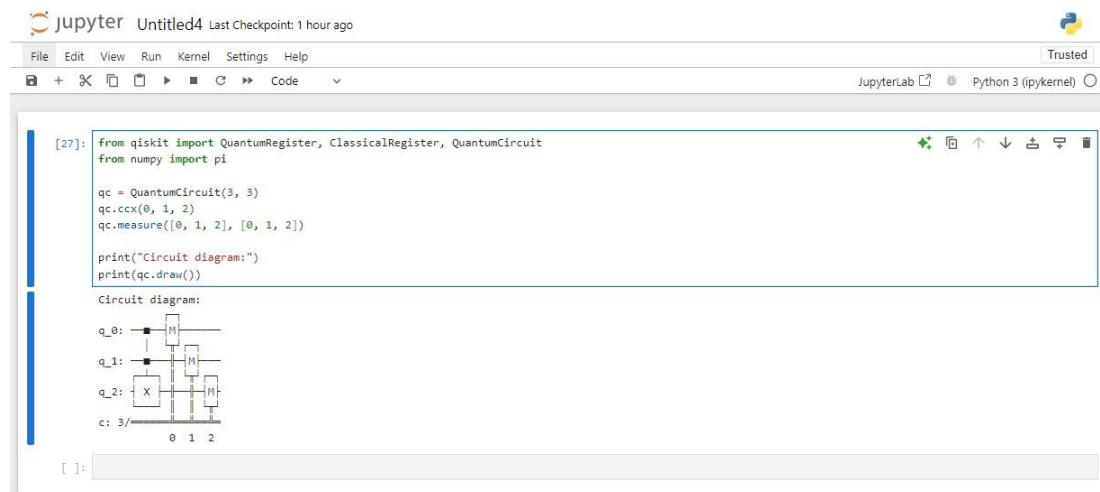
#### Program:

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

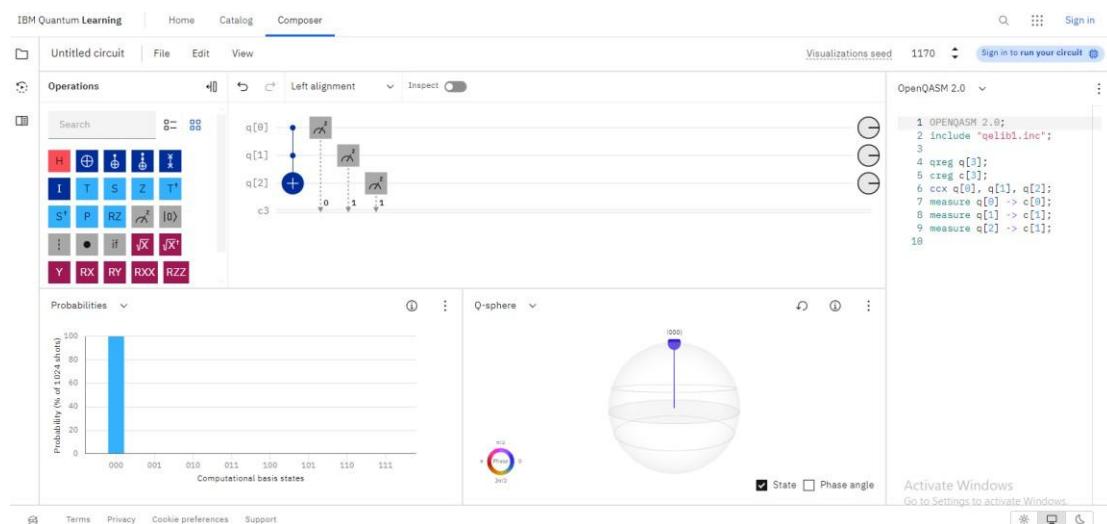
```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit  
from numpy import pi
```

```
qc = QuantumCircuit(3, 3)  
qc.ccx(0, 1, 2)  
qc.measure([0, 1, 2], [0, 1, 2])  
  
print("Circuit diagram:")  
print(qc.draw())
```

#### Output:



```
jupyter Untitled4 Last Checkpoint: 1 hour ago  
File Edit View Run Kernel Settings Help Trusted  
+ X Code JupyterLab Python 3 (ipykernel)  
[27]: 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())  
  
Circuit diagram:  
q_0: ┌───┐  
     └───┘  
q_1: ┌───┐  
     └───┘  
q_2: ┌──X──┐  
      └───┘  
c: 3/ 0 1 2
```



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())
```

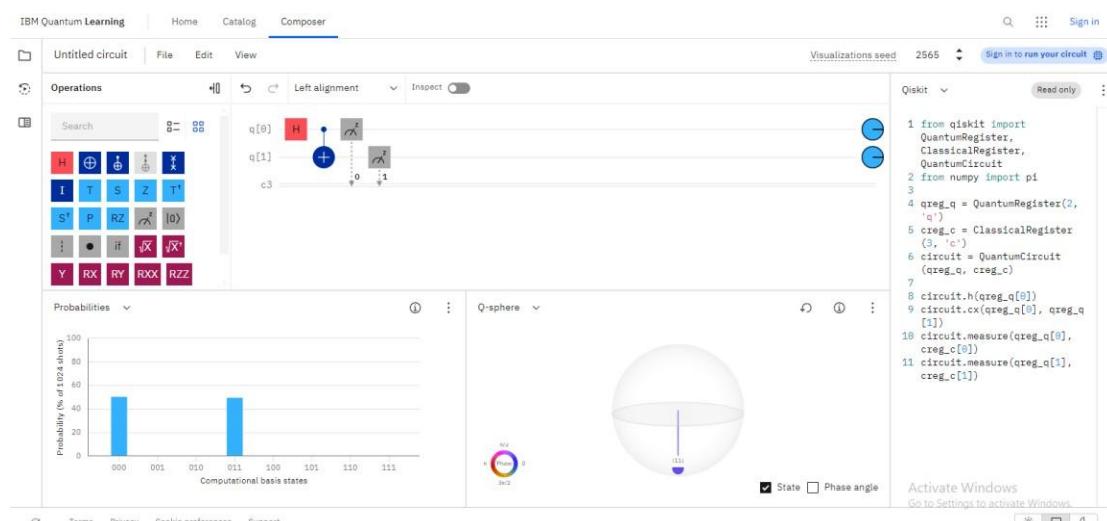
#### Output:

The screenshot shows a Jupyter Notebook cell with the following content:

```
*[41]: 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())
```

Below the code, the text "Circuit diagram:" is followed by a quantum circuit diagram:

```
Circuit diagram:  
q_0: ┌──H─────────┐  
      ┌──M─────────┐  
      ┌──M─────────┐  
q_1: └──X─────────┘  
      ┌──H─────────┐  
c: 2/ ───────────  
          0 1
```



## 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:

The screenshot shows a Jupyter Notebook cell with the following content:

```
[43]: 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())
```

Below the code, the output shows the "Circuit diagram:":

Circuit diagram:

The screenshot shows the IBM Quantum Learning interface with the following components:

- Operations Panel:** A sidebar containing a search bar and a grid of quantum gate icons: H, T, S, I, RZ, RX, RY, RXX, RYY, RZZ.
- Quantum Circuit Editor:** A main area where a quantum circuit is being built. It shows two qubits (q[0] and q[1]) and one classical register (c[0]). A CNOT gate is placed between q[0] and q[1].
- Probabilities Plot:** A bar chart titled "Probabilities" showing the probability distribution across computational basis states (000, 001, 010, 011, 100, 101, 110, 111). The x-axis is labeled "Computational basis states" and the y-axis is labeled "Probability (% of 1024 shots)" ranging from 0 to 100.
- Q-sphere:** A 3D visualization of the state vector on a unit sphere. The vector points along the z-axis.
- Code Editor:** A right-hand panel displaying the corresponding Python code for the circuit:

```
1 from qiskit import  
2 QuantumRegister,  
3 ClassicalRegister,  
4 QuantumCircuit  
5 from numpy import pi  
6 qreg_q = QuantumRegister(2,  
7 'q')  
8 creg_c = ClassicalRegister  
9 (3, 'c')  
10 circuit = QuantumCircuit  
11 (qreg_q, creg_c)  
12  
13 circuit.cx(qreg_q[0], qreg_q  
14 [1])  
15 circuit.h(qreg_q[0])  
16 circuit.measure(qreg_q[1],  
17 creg_c[1])  
18 circuit.measure(qreg_q[0],  
19 creg_c[0])
```

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
```

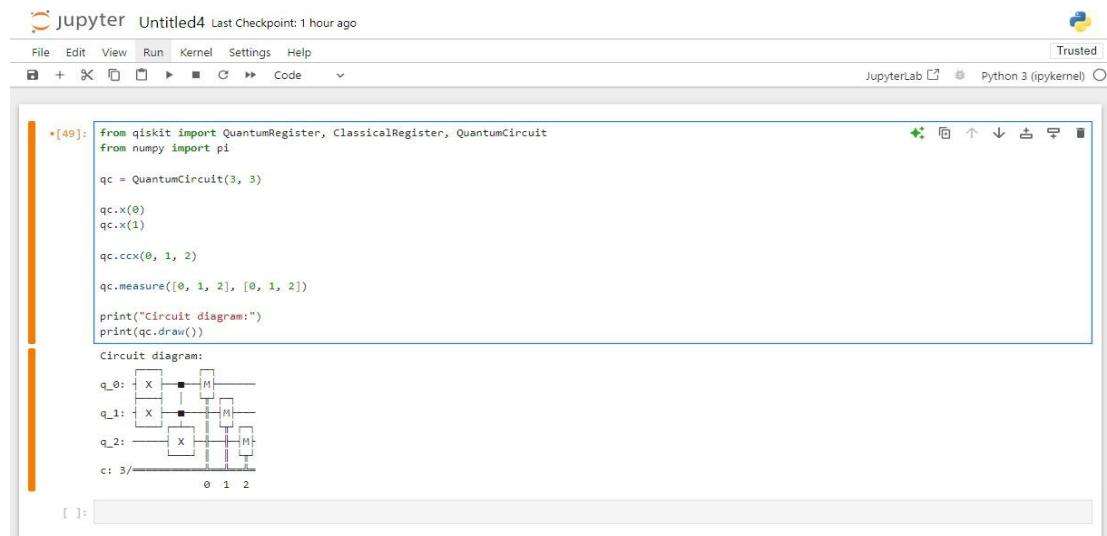
```
qc = QuantumCircuit(3, 3)
```

```
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())
```

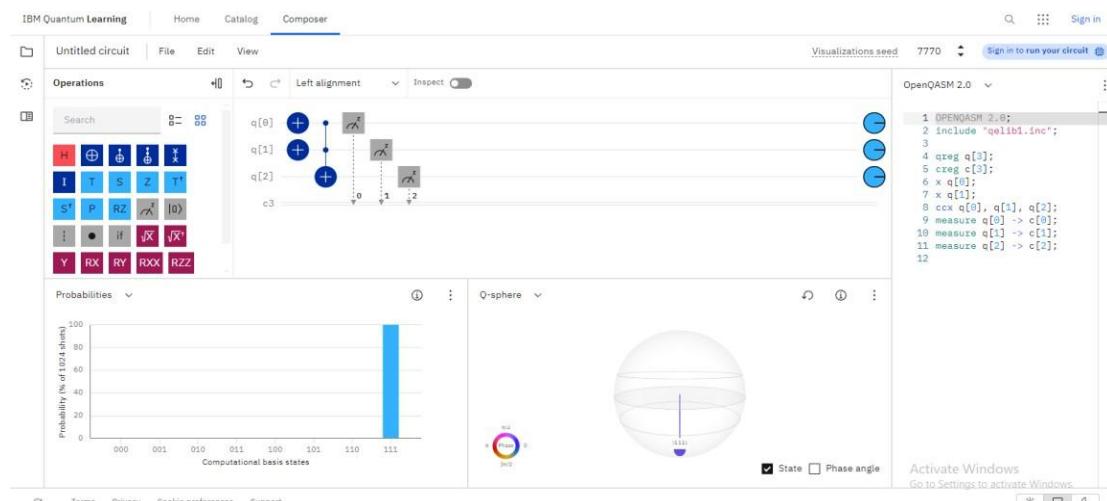
#### Output:



```
*[49]: 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())
```

Circuit diagram:

```
graph LR; q0((q_0)) -- X --> m0(( )); q1((q_1)) -- X --> m1(( )); q2((q_2)) -- X --> m2(( )); q0 --- ccx((CCX)); q1 --- ccx; q2 --- ccx; m0 --- c(( )); m1 --- c; m2 --- c;
```



IBM Quantum Learning | Home | Catalog | Composer

Visualizations seed: 7770 | Sign in to run your circuit

Operations

Probabilities

OpenQASM 2.0

```
1 OPENQASM 2.0;  
2 include "qelib1.inc";  
3  
4 qreg q[3];  
5 creg c[3];  
6 q[0];  
7 x q[0];  
8 ccx q[0], q[1], q[2];  
9 measure q[0] -> c[0];  
10 measure q[1] -> c[1];  
11 measure q[2] -> c[2];  
12
```