|  |
| --- |
|  |
| **Solution** :  def my\_circuit(theta, phi):  ##################  qml.CNOT(wires=[0, 1])  qml.Hadamard(wires=0)  qml.RX(theta, wires=2)  qml.CNOT(wires=[2, 0])  qml.RY(phi, wires=1)  ##################  # REORDER THESE 5 GATES TO MATCH THE CIRCUIT IN THE PICTURE  # This is the measurement; we return the probabilities of all possible output states  # You'll learn more about what types of measurements are available in a later node  return qml.probs(wires=[0, 1, 2]) |
| **Qiskit Program**:  import numpy as np  import random  from qiskit.quantum\_info import Statevector  import pennylane as qml  import matplotlib.pyplot as plt  dev\_unique\_wires = qml.device("default.qubit", wires=3)  @qml.qnode(dev)  def my\_circuit(theta, phi):  ##################  qml.CNOT(wires=[0, 1])  qml.Hadamard(wires=0)  qml.RX(theta, wires=2)  qml.CNOT(wires=[2, 0])  qml.RY(phi, wires=1)  ##################  # REORDER THESE 5 GATES TO MATCH THE CIRCUIT IN THE PICTURE  # This is the measurement; we return the probabilities of all possible output states  # You'll learn more about what types of measurements are available in a later node  return qml.probs(wires=[0, 1, 2])  theta = np.pi/2  phi = np.pi  circuit = qml.QNode(my\_circuit, dev\_unique\_wires)  qml.drawer.use\_style("pennylane")  probs = qml.draw\_mpl(circuit)(theta, phi)  plt.show()  **O/P:** |

|  |
| --- |
|  |
| **Solution:**  # This creates a device with three wires on which PennyLane can run computations  dev = qml.device("default.qubit", wires=3)  def my\_circuit(theta, phi, omega):  ##################  qml.RX(theta, wires=0)  qml.RY(phi, wires=1)  qml.RZ(omega, wires=2)  qml.CNOT(wires=[0, 1])  qml.CNOT(wires=[1, 2])  qml.CNOT(wires=[2, 0])  ##################  # IMPLEMENT THE CIRCUIT BY ADDING THE GATES  # Here are two examples, so you can see the format:  # qml.CNOT(wires=[0, 1])  # qml.RX(theta, wires=0)  return qml.probs(wires=[0, 1, 2])  # This creates a QNode, binding the function and device  my\_qnode = qml.QNode(my\_circuit, dev)  # We set up some values for the input parameters  theta, phi, omega = 0.1, 0.2, 0.3  # Now we can execute the QNode by calling it like we would a regular function  my\_qnode(theta, phi, omega) |
| **Qiskit Program**:  import numpy as np  import random  from qiskit.quantum\_info import Statevector  import pennylane as qml  import matplotlib.pyplot as plt  dev = qml.device("default.qubit", wires=3)  @qml.qnode(dev)  def my\_circuit(theta, phi, omega):  ##################  qml.RX(theta, wires=0)  qml.RY(phi, wires=1)  qml.RZ(omega, wires=2)  qml.CNOT(wires=[0, 1])  qml.CNOT(wires=[1, 2])  qml.CNOT(wires=[2, 0])  ##################  # IMPLEMENT THE CIRCUIT BY ADDING THE GATES  # Here are two examples, so you can see the format:  # qml.CNOT(wires=[0, 1])  # qml.RX(theta, wires=0)  return qml.probs(wires=[0, 1, 2])  # This creates a QNode, binding the function and device  my\_qnode = qml.QNode(my\_circuit, dev)  # We set up some values for the input parameters  theta, phi, omega = 0.1, 0.2, 0.3  # Now we can execute the QNode by calling it like we would a regular function  my\_qnode(theta, phi, omega)  qml.drawer.use\_style("pennylane")  probs = qml.draw\_mpl(my\_qnode)(theta, phi,omega)  plt.show()  **O/P:** |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=3)  # DECORATE THE FUNCTION BELOW TO TURN IT INTO A QNODE  @qml.qnode(dev)  def my\_circuit(theta, phi, omega):  qml.RX(theta, wires=0)  qml.RY(phi, wires=1)  qml.RZ(omega, wires=2)  qml.CNOT(wires=[0, 1])  qml.CNOT(wires=[1, 2])  qml.CNOT(wires=[2, 0])  return qml.probs(wires=[0, 1, 2])  theta, phi, omega = 0.1, 0.2, 0.3  ##################  circuit = qml.QNode(my\_circuit, dev)  result = circuit(theta, phi,omega)  ##################  # RUN THE QNODE WITH THE PROVIDED PARAMETERS |
| **Qiskit Program**:  import numpy as np  import random  from qiskit.quantum\_info import Statevector  import pennylane as qml  import matplotlib.pyplot as plt  dev = qml.device("default.qubit", wires=3)  @qml.qnode(dev)  def my\_circuit(theta, phi, omega):  ##################  qml.RX(theta, wires=0)  qml.RY(phi, wires=1)  qml.RZ(omega, wires=2)  qml.CNOT(wires=[0, 1])  qml.CNOT(wires=[1, 2])  qml.CNOT(wires=[2, 0])  ##################  # IMPLEMENT THE CIRCUIT BY ADDING THE GATES  # Here are two examples, so you can see the format:  # qml.CNOT(wires=[0, 1])  # qml.RX(theta, wires=0)  return qml.probs(wires=[0, 1, 2])  # We set up some values for the input parameters  theta, phi, omega = 0.1, 0.2, 0.3  circuit = qml.QNode(my\_circuit, dev)  qml.drawer.use\_style("pennylane")  result = qml.draw\_mpl(circuit)(theta, phi,omega)  plt.show()  **O/P** |

|  |
| --- |
|  |
| **Solution**: |
| **Qiskit Program:**  import numpy as np  import random  from qiskit.quantum\_info import Statevector  import pennylane as qml  import matplotlib.pyplot as plt  dev = qml.device("default.qubit", wires=3)  @qml.qnode(dev)  def my\_circuit(theta, phi, omega):  ##################  qml.RX(theta, wires=0)  qml.RY(phi, wires=1)  qml.RZ(omega, wires=2)  qml.CNOT(wires=[0, 1])  qml.CNOT(wires=[1, 2])  qml.CNOT(wires=[2, 0])  ##################  # IMPLEMENT THE CIRCUIT BY ADDING THE GATES  # Here are two examples, so you can see the format:  # qml.CNOT(wires=[0, 1])  # qml.RX(theta, wires=0)  return qml.probs(wires=[0, 1, 2])  # We set up some values for the input parameters  theta, phi, omega = 0.1, 0.2, 0.3  circuit = qml.QNode(my\_circuit, dev)  specs\_func = qml.specs(circuit)  specs\_func(theta, phi,omega) |