|  |  |
| --- | --- |
|  | |
| **Solution** :  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_rx\_pi(state):  """Apply an RX gate with an angle of \pi to a particular basis state.  Args:  state (int): Either 0 or 1. If 1, initialize the qubit to state |1>  before applying other operations.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  if state == 1:  qml.PauliX(wires=0)  ##################  qml.RX(np.pi,wires=0)  ##################  # APPLY RX(pi) AND RETURN THE STATE  return qml.state()  print(apply\_rx\_pi(0))  print(apply\_rx\_pi(1)) | |
| **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=1)  @qml.qnode(dev)  @qml.qnode(dev)  def apply\_rx\_pi(state):  """Apply an RX gate with an angle of \pi to a particular basis state.  Args:  state (int): Either 0 or 1. If 1, initialize the qubit to state |1>  before applying other operations.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  if state == 1:  qml.PauliX(wires=0)  ##################  # APPLY RX(pi) AND RETURN THE STATE  qml.RX(np.pi,wires=0)  ##################    return qml.state()    print(apply\_rx\_pi(0))  print(apply\_rx\_pi(1))  **O/P:** |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_rx(theta, state):  """Apply an RX gate with an angle of theta to a particular basis state.  Args:  theta (float): A rotation angle.  state (int): Either 0 or 1. If 1, initialize the qubit to state |1>  before applying other operations.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  if state == 1:  qml.PauliX(wires=0)  ##################  # APPLY RX(theta) AND RETURN THE STATE  qml.RX(theta,wires=0)  ##################  # APPLY RX(theta) AND RETURN THE STATE  return qml.state()  # Code for plotting  angles = np.linspace(0, 4 \* np.pi, 200)  output\_states = np.array([apply\_rx(t, 0) for t in angles])  plot = plotter(angles, output\_states) |
| **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=1)  @qml.qnode(dev)  def apply\_rx(theta, state):  """Apply an RX gate with an angle of theta to a particular basis state.  Args:  theta (float): A rotation angle.  state (int): Either 0 or 1. If 1, initialize the qubit to state |1>  before applying other operations.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  if state == 1:  qml.PauliX(wires=0)  ##################  # APPLY RX(theta) AND RETURN THE STATE  qml.RX(theta,wires=0)  ##################  return qml.state()  # Code for plotting  angles = np.linspace(0, 4 \* np.pi, 200)  output\_states = np.array([apply\_rx(t, 0) for t in angles])  O/P: |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_ry(theta, state):  """Apply an RY gate with an angle of theta to a particular basis state.  Args:  theta (float): A rotation angle.  state (int): Either 0 or 1. If 1, initialize the qubit to state |1>  before applying other operations.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  if state == 1:  qml.PauliX(wires=0)  ##################  qml.RY(theta,wires=0)  ##################  # APPLY RY(theta) AND RETURN THE STATE  return qml.state()  # Code for plotting  angles = np.linspace(0, 4 \* np.pi, 200)  output\_states = np.array([apply\_ry(t, 0) for t in angles])  plot = plotter(angles, output\_states) |
| **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=1)  @qml.qnode(dev)  def apply\_ry(theta, state):  """Apply an RX gate with an angle of theta to a particular basis state.  Args:  theta (float): A rotation angle.  state (int): Either 0 or 1. If 1, initialize the qubit to state |1>  before applying other operations.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  if state == 1:  qml.PauliX(wires=0)  ##################  # APPLY RX(theta) AND RETURN THE STATE  qml.RY(theta,wires=0)  ##################  return qml.state()  # Code for plotting  angles = np.linspace(0, 4 \* np.pi, 200)  output\_states = np.array([apply\_ry(t, 0) for t in angles])  #plot = plotter(angles, output\_states)  **O/P**: |