|  |  |
| --- | --- |
|  | |
| **Solution** :  dev = qml.device("default.qubit", wires=1)  ##################  ket\_0 = np.array([1, 0])  target\_element\_magnitude = 1/np.sqrt(2)  U = np.array([[1, 1], [1, -1]]) / np.sqrt(2) # H Matrix  result\_state = np.dot(U,ket\_0)  print("result\_state:",result\_state)  sin\_theta\_2 = target\_element\_magnitude  theta = 2 \* np.arcsin(sin\_theta\_2)  print("theta:",theta)  cos\_theta\_2 = np.cos(theta / 2)  print("cos\_theta\_2:",cos\_theta\_2)  sin\_theta\_2 = np.sin(theta / 2)  print("sin\_theta\_2:",sin\_theta\_2)  # Constructing the RX gate matrix  RX = np.array([  [cos\_theta\_2, -1j \* sin\_theta\_2],  [-1j \* sin\_theta\_2, cos\_theta\_2]  ])  # ADJUST THE VALUES OF PHI, THETA, AND OMEGA  cos\_theta\_2 = np.pi / 2  phi, theta, omega = cos\_theta\_2, cos\_theta\_2, cos\_theta\_2  ##################  @qml.qnode(dev)  def hadamard\_with\_rz\_rx():  qml.RZ(phi, wires=0)  qml.RX(theta, wires=0)  qml.RZ(omega, wires=0)  return qml.state() | |
| **Qiskit Program**:  import numpy as np  import random  import pennylane as qml  import matplotlib.pyplot as plt  dev = qml.device("default.qubit", wires=1)  def get\_angle\_based\_on\_val(desired\_value):  # Find angles theta such that cos(theta) = desired\_value  theta\_cos = np.arccos(desired\_value)  theta\_deg\_cos = np.degrees(theta\_cos)    # Find angles theta such that sin(theta) = desired\_value  theta\_sin = np.arcsin(desired\_value)  theta\_deg\_sin = np.degrees(theta\_sin)    print(f"Angle (cosine) theta = {theta\_deg\_cos} degrees or {theta\_cos} radians")  print(f"Angle (sine) theta = {theta\_deg\_sin} degrees or {theta\_sin} radians")  return theta\_deg\_cos,theta\_deg\_sin    @qml.qnode(dev)  def hadamard\_with\_rz\_rx():  qml.RZ(phi, wires=0)  qml.RX(theta, wires=0)  qml.RZ(omega, wires=0)  return qml.state()    ##################  ket\_0 = np.array([1, 0])  target\_element\_magnitude = 1/np.sqrt(2) # Coming from Hadamard  U = np.array([[1, 1], [1, -1]]) / np.sqrt(2) # H Matrix  result\_state = np.dot(U,ket\_0)  print("result\_state:",result\_state)  sin\_theta\_2 = target\_element\_magnitude  theta = 2 \* np.arcsin(sin\_theta\_2)  print("theta:",theta)  print("get\_angle\_based\_on\_val",get\_angle\_based\_on\_val(sin\_theta\_2))  cos\_theta\_2 = np.cos(theta / 2)  print("cos\_theta\_2:",cos\_theta\_2)  sin\_theta\_2 = np.sin(theta / 2)  print("sin\_theta\_2:",sin\_theta\_2)  # Constructing the RX gate matrix - just for fun sake :)  RX = np.array([  [cos\_theta\_2, -1j \* sin\_theta\_2],  [-1j \* sin\_theta\_2, cos\_theta\_2]  ])  print("RX:",RX)  # ADJUST THE VALUES OF PHI, THETA, AND OMEGA  cos\_theta\_2 = np.pi / 2  print("cos\_theta\_2:",cos\_theta\_2)  phi, theta, omega = cos\_theta\_2, cos\_theta\_2, cos\_theta\_2  state\_to\_match = np.array(hadamard\_with\_rz\_rx())  print("state\_to\_match:",state\_to\_match)  inner\_product = np.vdot(state\_to\_match, result\_state) # inner product <state1|state2>  print("inner\_product:",inner\_product)  # Check if the states are identical  identical = np.isclose(np.abs(inner\_product), 1.0)  print(f"The states are identical: {identical}")  ##################  **O/P:** |

|  |
| --- |
|  |
| **Solution:** |
| **Qiskit Program**: |