|  |  |
| --- | --- |
|  | |
| **Solution** :  dev = qml.device("default.qubit", wires=1)  U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)  @qml.qnode(dev)  def varied\_initial\_state(state):  """Complete the function such that we can apply the operation U to  either |0> or |1> depending on the input argument flag.  Args:  state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,  otherwise, leave it in state 0.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  if state > 0:  qml.PauliX(wires=0)  ##################  # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER  ##################  # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER  # APPLY U TO THE STATE  qml.QubitUnitary(U, wires=0)  return qml.state() | |
| **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)  U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)  @qml.qnode(dev)  def varied\_initial\_state(state):  """Complete the function such that we can apply the operation U to  either |0> or |1> depending on the input argument flag.  Args:  state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,  otherwise, leave it in state 0.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  if state > 0:  qml.PauliX(wires=0)  ##################  # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER  # APPLY U TO THE STATE  qml.QubitUnitary(U, wires=0)  return qml.state()  state = 0 # can be 1 or 0  circuit = qml.QNode(varied\_initial\_state, dev)  print(circuit(state))  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)  U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)  @qml.qnode(dev)  def apply\_u():  ##################  qml.QubitUnitary(U, wires=0)  ##################  # USE QubitUnitary TO APPLY U TO THE QUBIT  # Return the state  return qml.state() |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_hadamard():  ##################  qml.Hadamard(wires=0)  ##################  # APPLY THE HADAMARD GATE  # RETURN THE STATE  return qml.state() |
| **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\_hadamard():  ##################  qml.Hadamard(wires=0)  ##################  # APPLY THE HADAMARD GATE  # RETURN THE STATE  return qml.state()    circuit = qml.QNode(apply\_hadamard, dev)  print(circuit())  **O/P**: |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_hadamard\_to\_state(state):  """Complete the function such that we can apply the Hadamard to  either |0> or |1> depending on the input argument flag.  Args:  state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,  otherwise, leave it in state 0.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  if state > 0:  qml.PauliX(wires=0)  ##################  # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON state  # APPLY THE HADAMARD  qml.Hadamard(wires=0)  # RETURN THE STATE  return qml.state()  print(apply\_hadamard\_to\_state(0))  print(apply\_hadamard\_to\_state(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)  def apply\_hadamard\_to\_state(state):  """Complete the function such that we can apply the Hadamard to  either |0> or |1> depending on the input argument flag.  Args:  state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,  otherwise, leave it in state 0.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  if state > 0:  qml.PauliX(wires=0)  ##################  # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON state  # APPLY THE HADAMARD  qml.Hadamard(wires=0)    # RETURN THE STATE  return qml.state()    print(apply\_hadamard\_to\_state(0))  print(apply\_hadamard\_to\_state(1))  **O/P:** |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_hxh(state):  """Complete the function such that we can apply the Hadamard to  either |0> or |1> depending on the input argument flag.  Args:  state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,  otherwise, leave it in state 0.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  if state > 0 :  qml.PauliX(wires=0)  qml.Hadamard(wires=0)  qml.PauliX(wires=0)  qml.Hadamard(wires=0)  ##################  # RETURN THE STATE  return qml.state()  # Print your results  print(apply\_hxh(0))  print(apply\_hxh(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)  def apply\_hxh(state):  """Complete the function such that we can apply the Hadamard to  either |0> or |1> depending on the input argument flag.  Args:  state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,  otherwise, leave it in state 0.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  if state > 0 :  qml.PauliX(wires=0)    qml.Hadamard(wires=0)  qml.PauliX(wires=0)  qml.Hadamard(wires=0)    ##################  # RETURN THE STATE  return qml.state()    # Print your results  print(apply\_hxh(0))  print(apply\_hxh(1))  **O/P:** |