|  |  |
| --- | --- |
|  | |
| **Solution** :  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def apply\_z\_to\_plus():  """Write a circuit that applies PauliZ to the |+> state and returns  the state.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  qml.Hadamard(wires=0)  ##################  # CREATE THE |+> STATE  # APPLY PAULI Z  qml.PauliZ(wires=0)  # RETURN THE STATE  return qml.state()  print(apply\_z\_to\_plus()) | |
| **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\_z\_to\_plus():  """Write a circuit that applies PauliZ to the |+> state and returns  the state.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  qml.Hadamard(wires=0)  ##################  # CREATE THE |+> STATE  # APPLY PAULI Z  qml.PauliZ(wires=0)    # RETURN THE STATE  return qml.state()  print(apply\_z\_to\_plus())  **O/P**: |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def fake\_z():  """Use RZ to produce the same action as Pauli Z on the |+> state.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  # CREATE THE |+> STATE  qml.Hadamard(wires=0)  ##################  angle = np.pi  # APPLY RZ  qml.RZ(angle,wires=0)    # 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 fake\_z():  """Use RZ to produce the same action as Pauli Z on the |+> state.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  # CREATE THE |+> STATE  qml.Hadamard(wires=0)  ##################  # APPLY RZ  qml.RZ(np.pi,wires=0)    # RETURN THE STATE  return qml.state()  print(fake\_z())  **O/P**: |

|  |
| --- |
|  |
| **Solution:**  dev = qml.device("default.qubit", wires=1)  @qml.qnode(dev)  def many\_rotations():  """Implement the circuit depicted above and return the quantum state.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  # CREATE THE |+> STATE  qml.Hadamard(wires=0)  ##################  # IMPLEMENT THE CIRCUIT  qml.S(wires=0)  qml.adjoint(qml.T)(wires=0)  qml.RZ(0.3,wires=0)  qml.adjoint(qml.S)(wires=0)  # 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 many\_rotations():  """Implement the circuit depicted above and return the quantum state.  Returns:  np.array[complex]: The state of the qubit after the operations.  """  ##################  # CREATE THE |+> STATE  qml.Hadamard(wires=0)  ##################  # IMPLEMENT THE CIRCUIT  qml.S(wires=0)  qml.adjoint(qml.T)(wires=0)  qml.RZ(0.3,wires=0)  qml.adjoint(qml.S)(wires=0)  # RETURN THE STATE  return qml.state()  print(many\_rotations())  O/P: |