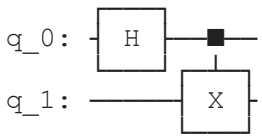
PRÁCTICA 4. "QUANTUM TELEPORTATION

MIGUEL ÁNGEL NAVARRO ARENAS.

```
!pip install qiskit
!pip install git+https://github.com/qiskit-community/qiskit-textbook.git#subdirecto
!pip install numexpr
!pip install pylatexenc

# Do the necessary imports
import numpy as np
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import IBMQ, Aer, transpile, assemble, execute
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from qiskit.extensions import Initialize
from qiskit_textbook.tools import random_state, array_to_latex
from math import sqrt, pi
from qiskit.quantum_info import *
from qiskit.visualization import *
from qiskit.result import *
import math
```

1.- CREATING BELL PAIRS

```
qc_Bellpair = QuantumCircuit(2)
#### your code goes here

qc_Bellpair.h(0)
qc_Bellpair.cnot(0,1)


# Let's view our circuit
qc_Bellpair.draw()
```



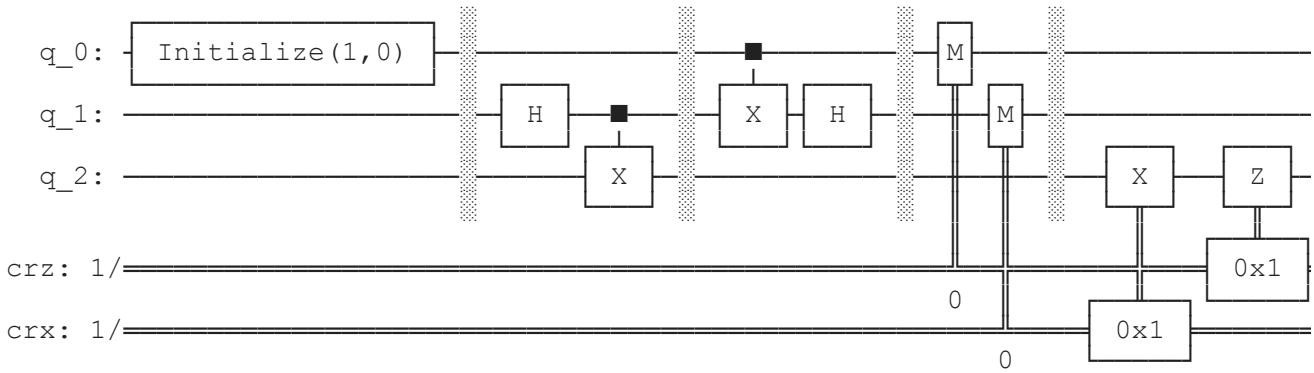

```
backend = Aer.get_backend('statevector_simulator') # Tell Qiskit how to simulate ou
# Let's display the output state vector
result = execute(qc_Bellpair,backend).result() # Do the simulation, returning the r
out_state = result.get_statevector()
print(out_state) # Display the output state vector
```

```
Statevector([0.70710678+0.j, 0.        +0.j, 0.        +0.j,
             0.70710678+0.j],
            dims=(2, 2))
```

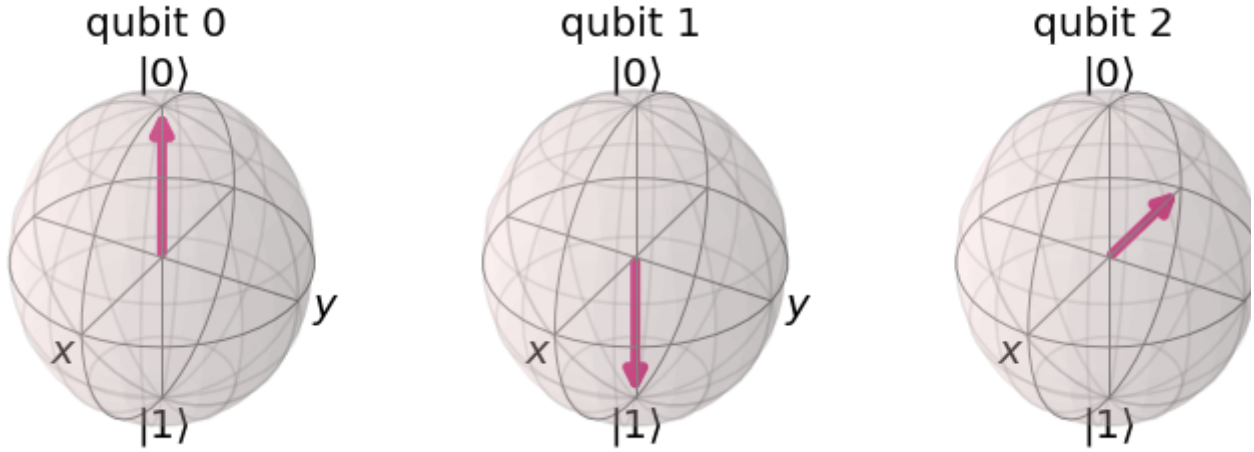
2.- TRANSFERING QUANTUM STATES: THE TELEPORTATION PROTOCOL

```
def init_state(qc, a):

    #### your code goes here

    desired_vector=[1,0]
    qc.initialize(desired_vector,a)
    qc.draw()
    qc.barrier()


def create_bell_pair(qc, a, b):
    """Creates a bell pair in qc using qubits a & b"""
    #### your code goes here
    qc.h(a)
    qc.cx(a,b)
    qc.draw()


def alice_gates(qc, psi, a):
    #### your code goes here
    qc.cx(psi,a)
    qc.h(a)


def measure_and_send(qc, a, b):
    """Measures qubits a & b and 'sends' the results to Bob"""
    qc.barrier()
    #### your code goes here
    qc.measure(a,0)
    qc.measure(b,1)


# This function takes a QuantumCircuit (qc), integer (qubit)
# and ClassicalRegisters (crz & crx) to decide which gates to apply
def bob_gates(qc, qubit, crz, crx):
    #### your code goes here
    qc.x(qubit).c_if(crx,1)
    qc.z(qubit).c_if(crz,1)


## SETUP
# Protocol uses 3 qubits and 2 classical bits in 2 different registers
qr = QuantumRegister(3, name="q")
crz, crx = ClassicalRegister(1, name="crz"), ClassicalRegister(1, name="crx")
teleportation_circuit = QuantumCircuit(qr, crz, crx)

## STEP 1
#### your code goes here
```

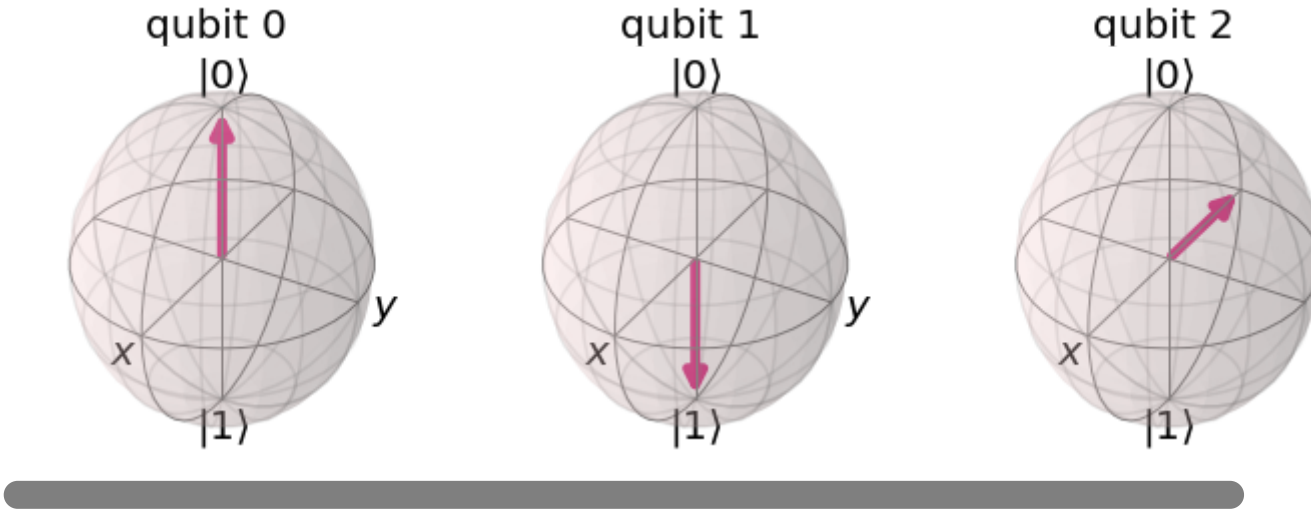
```
init_state(teleportation_circuit,0)



## STEP 2
#### your code goes here
create_bell_pair(teleportation_circuit,1,2)


## STEP 3 & 4
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
alice_gates(teleportation_circuit,0,1)
measure_and_send(teleportation_circuit,0,1)


## STEP 5
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
bob_gates(teleportation_circuit,2,crz,crx)


teleportation_circuit.draw()
```



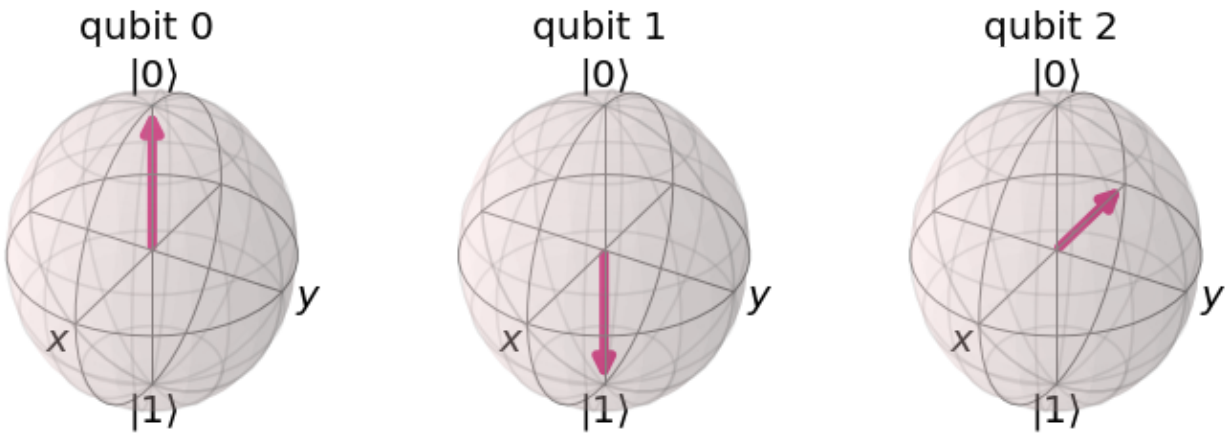

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



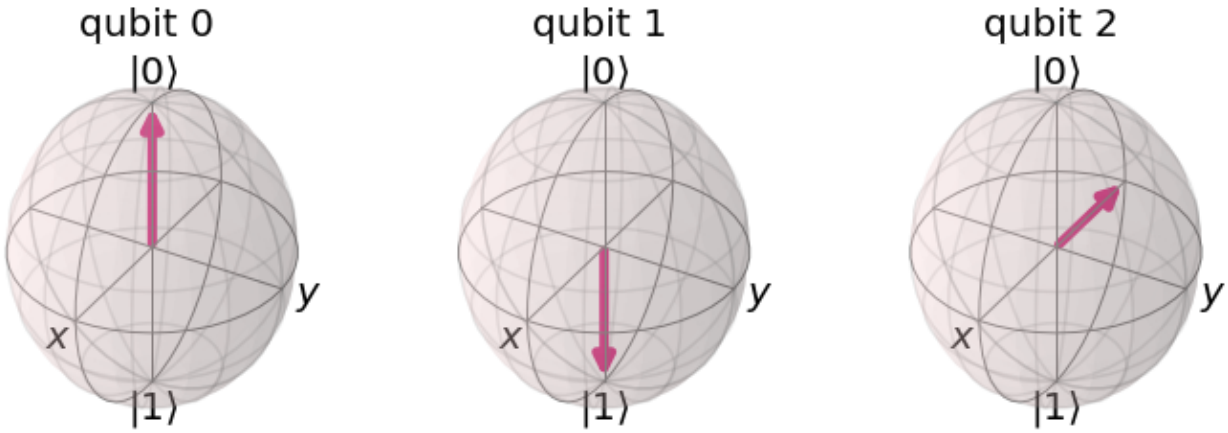
```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



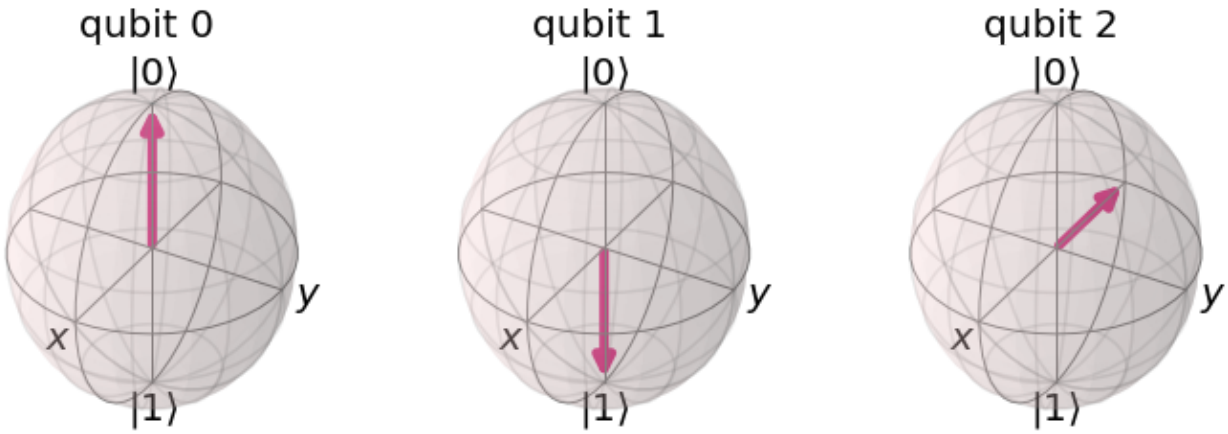
```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

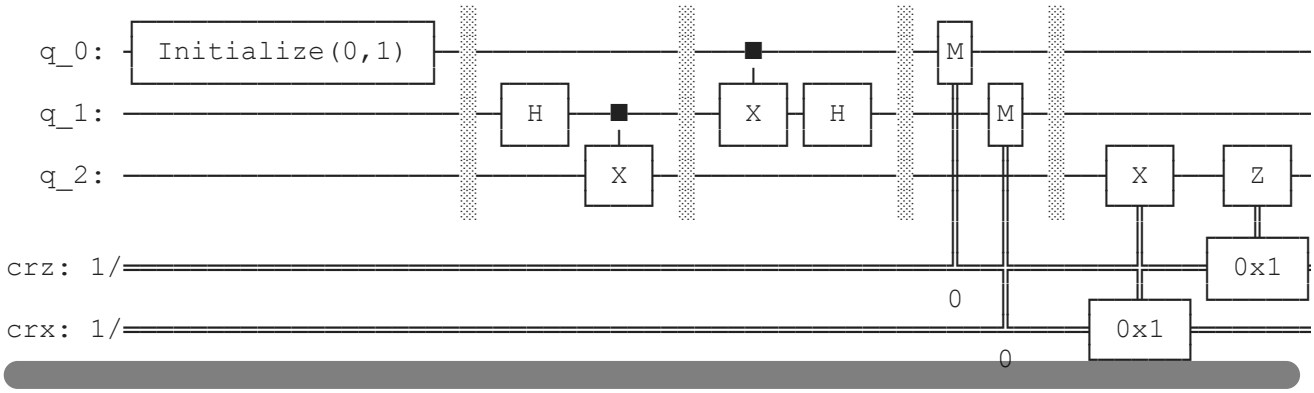



```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

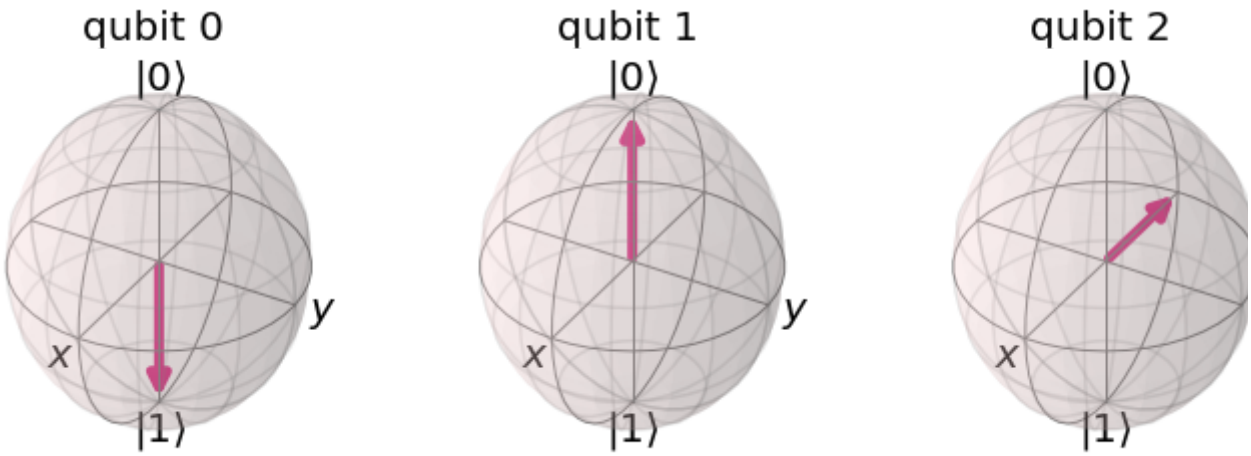



```
def init_state(qc, a):

    #### your code goes here

    desired_vector=[0,1]
    qc.initialize(desired_vector,a)
    qc.draw()
    qc.barrier()


## SETUP 2
# Protocol uses 3 qubits and 2 classical bits in 2 different registers
qr = QuantumRegister(3, name="q")
crz, crx = ClassicalRegister(1, name="crz"), ClassicalRegister(1, name="crx")
teleportation_circuit = QuantumCircuit(qr, crz, crx)

## STEP 1
```

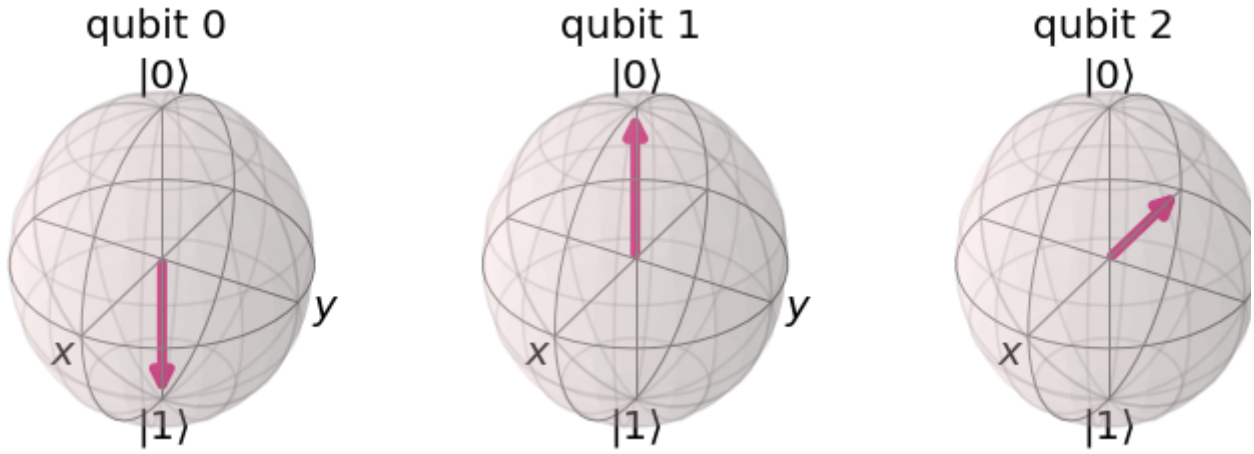
```
#### your code goes here
init_state(teleportation_circuit,0)



## STEP 2
#### your code goes here
create_bell_pair(teleportation_circuit,1,2)


## STEP 3 & 4
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
alice_gates(teleportation_circuit,0,1)
measure_and_send(teleportation_circuit,0,1)


## STEP 5
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
bob_gates(teleportation_circuit,2,crz,crx)


teleportation_circuit.draw()
```



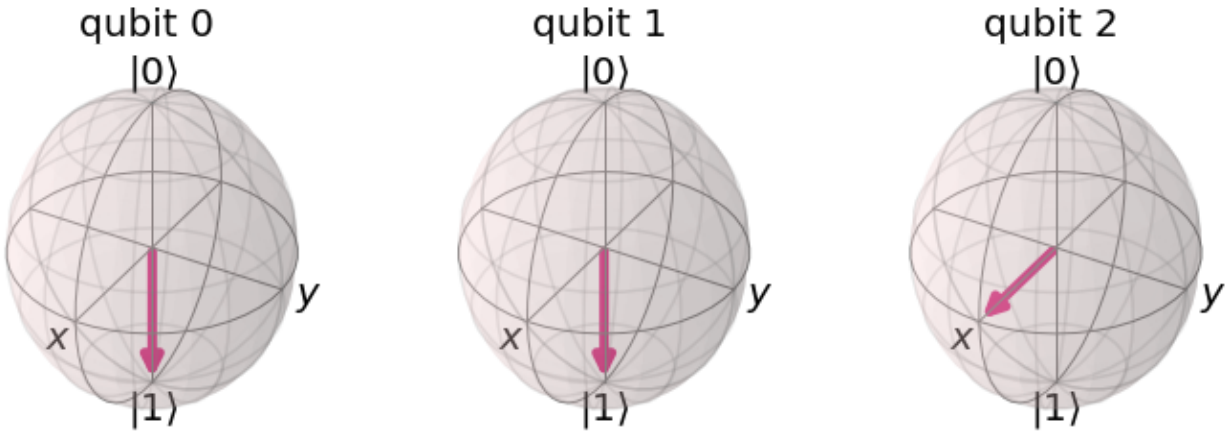

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



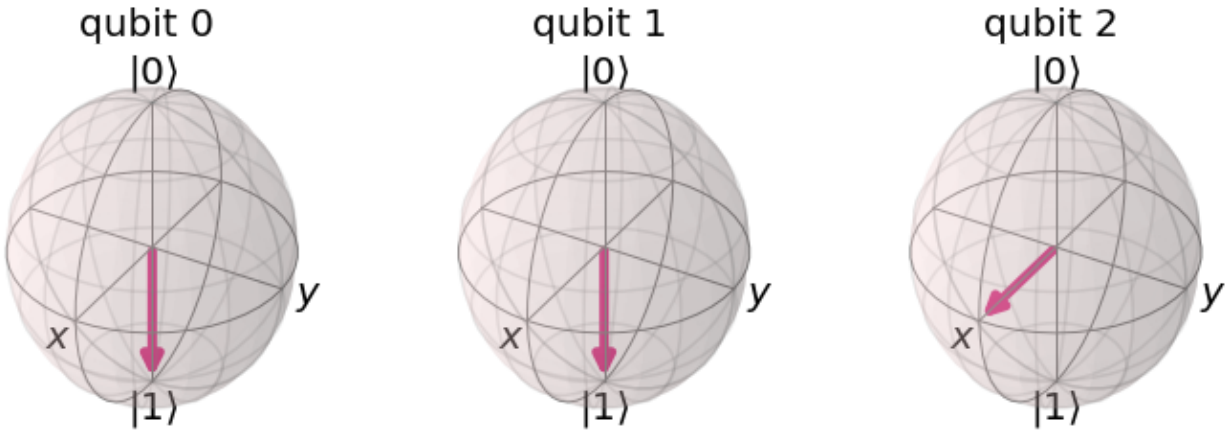
```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



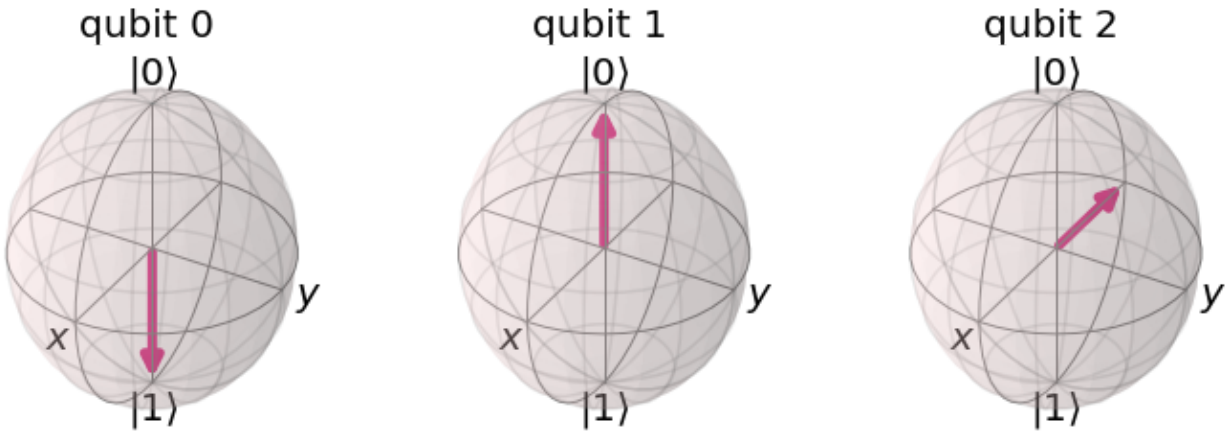

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```




```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

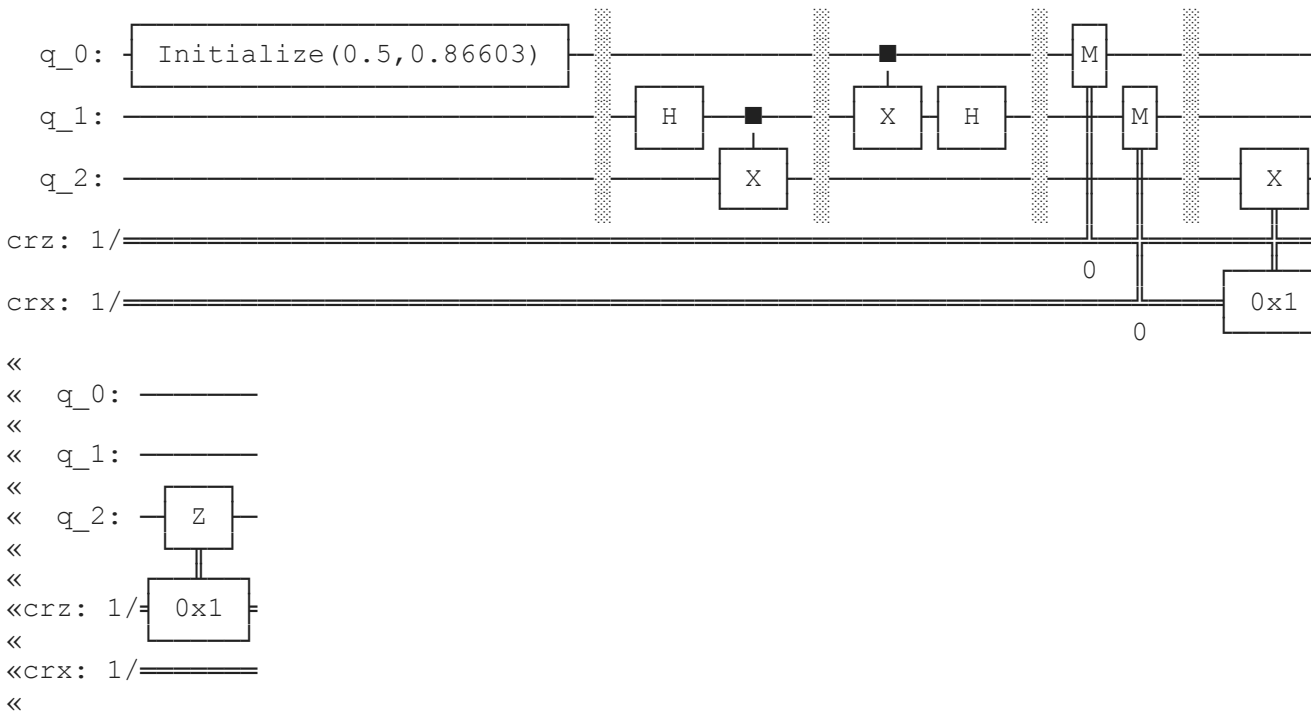


```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

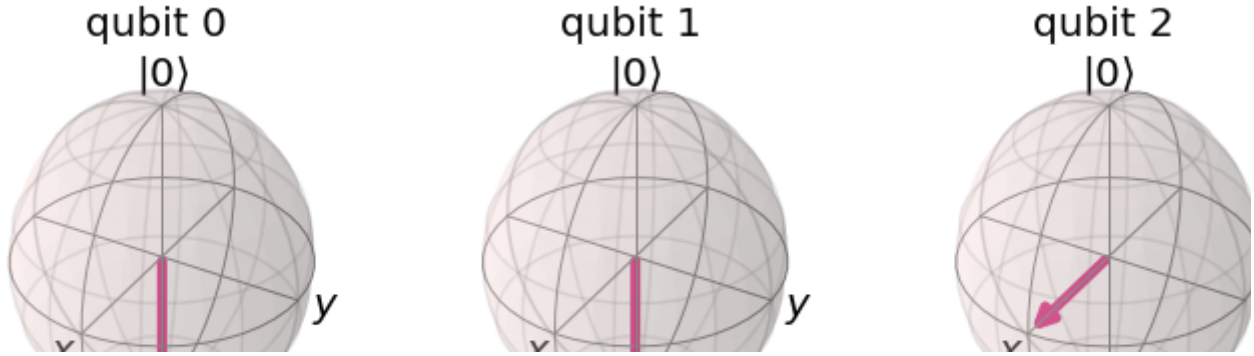


```
def init_state(qc, a):

    #### your code goes here

    desired_vector=[1/2,math.sqrt(3)/2]
    qc.initialize(desired_vector,a)
    qc.draw()
    qc.barrier()


## SETUP 3
# Protocol uses 3 qubits and 2 classical bits in 2 different registers
qr = QuantumRegister(3, name="q")
crz, crx = ClassicalRegister(1, name="crz"), ClassicalRegister(1, name="crx")
teleportation_circuit = QuantumCircuit(qr, crz, crx)
```

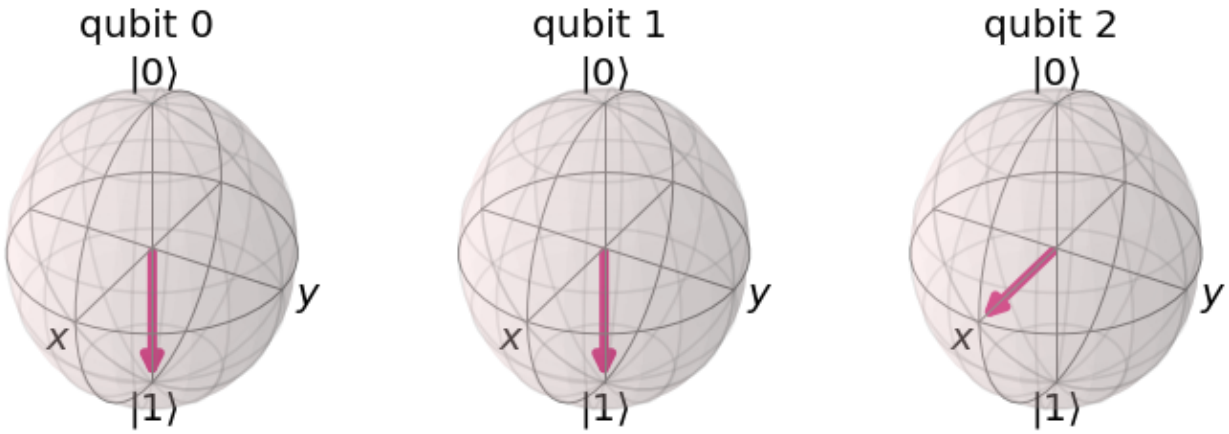
```
## STEP 1
#### your code goes here
init_state(teleportation_circuit,0)



## STEP 2
#### your code goes here
create_bell_pair(teleportation_circuit,1,2)


## STEP 3 & 4
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
alice_gates(teleportation_circuit,0,1)
measure_and_send(teleportation_circuit,0,1)


## STEP 5
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
bob_gates(teleportation_circuit,2,crz,crx)


teleportation_circuit.draw()
```



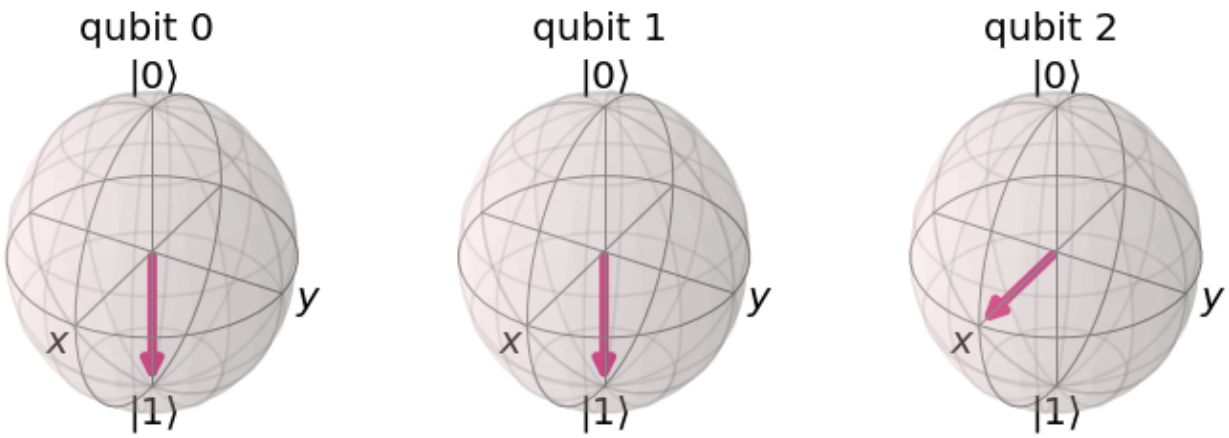

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



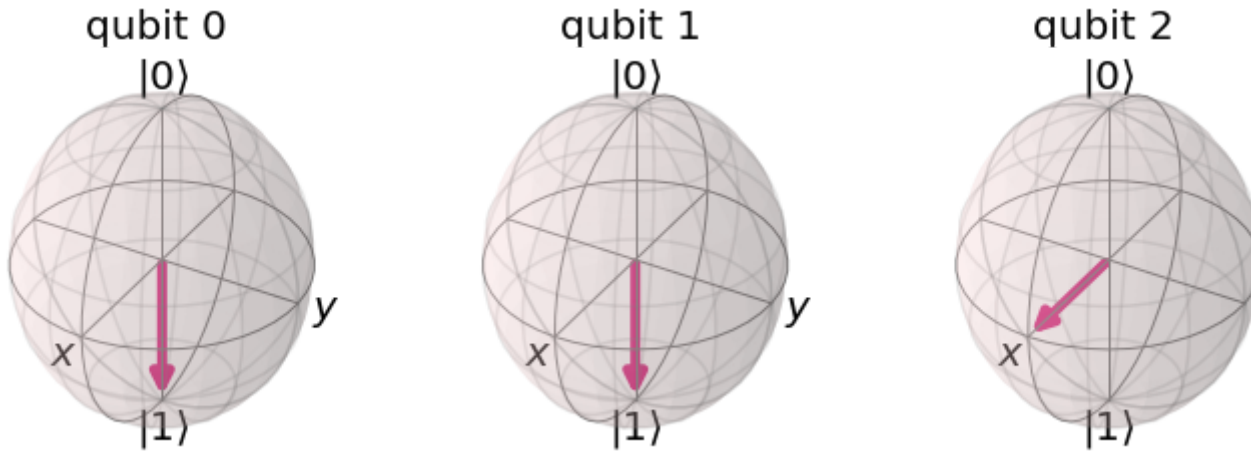

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



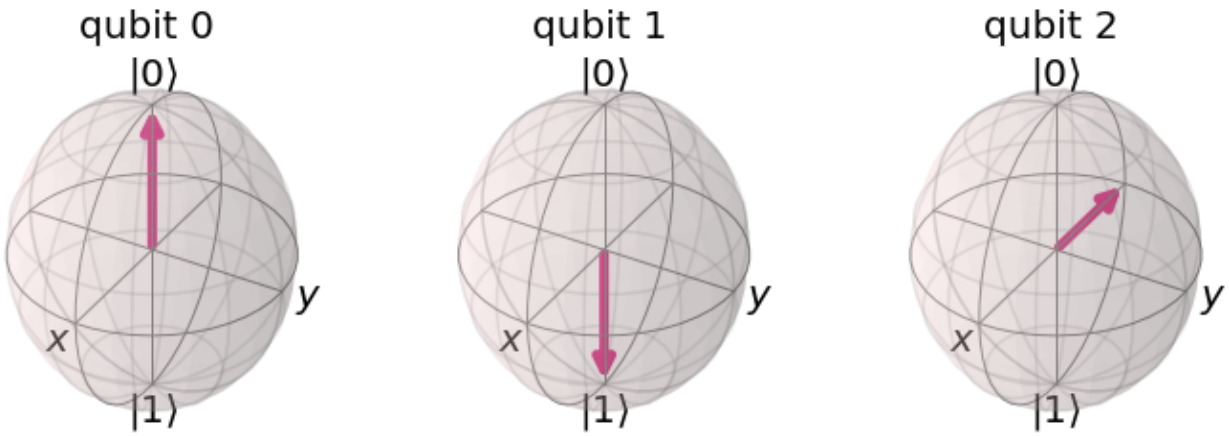

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```




```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



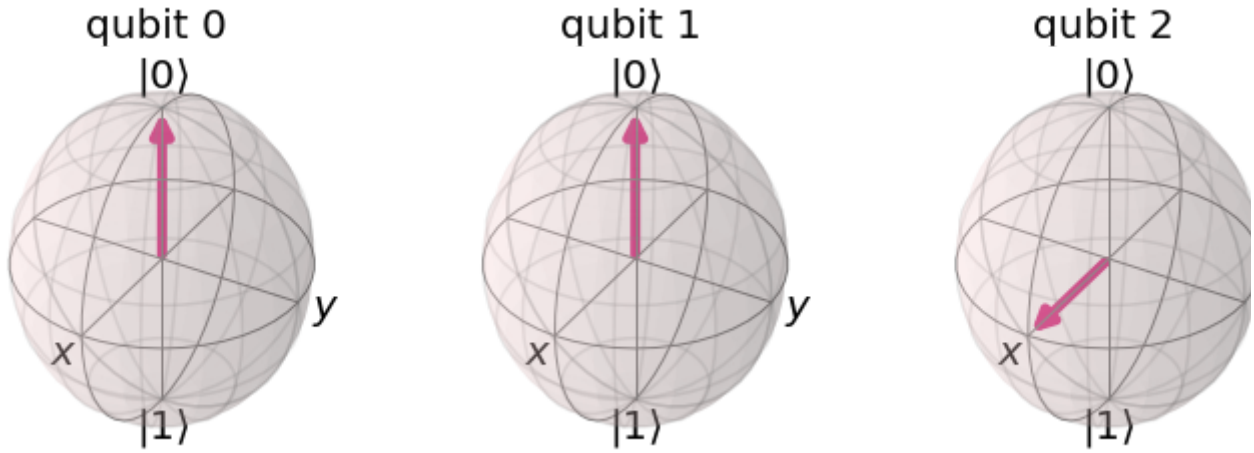

```
from qiskit.quantum_info import *
def init_state(qc, a):

    #### your code goes here
    rs = random_state(1)
    qc.initialize(rs,a)
```

```
    qc.draw()
    qc.barrier()
```

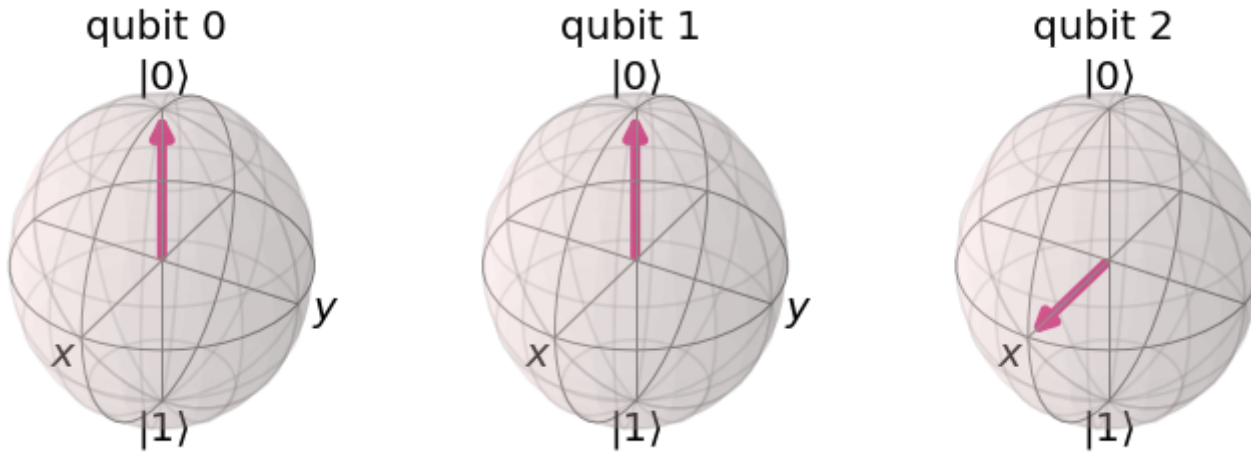
```
## SETUP 4
# Protocol uses 3 qubits and 2 classical bits in 2 different registers
qr = QuantumRegister(3, name="q")
crz, crx = ClassicalRegister(1, name="crz"), ClassicalRegister(1, name="crx")
teleportation_circuit = QuantumCircuit(qr, crz, crx)

## STEP 1
#### your code goes here
init_state(teleportation_circuit,0)



## STEP 2
#### your code goes here
create_bell_pair(teleportation_circuit,1,2)


## STEP 3 & 4
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
alice_gates(teleportation_circuit,0,1)
measure_and_send(teleportation_circuit,0,1)


## STEP 5
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
bob_gates(teleportation_circuit,2,crz,crx)


teleportation_circuit.draw()
```

q_0: ┤ Initialize(0.70482-0.62911j,0.28889-0.15486j) ├

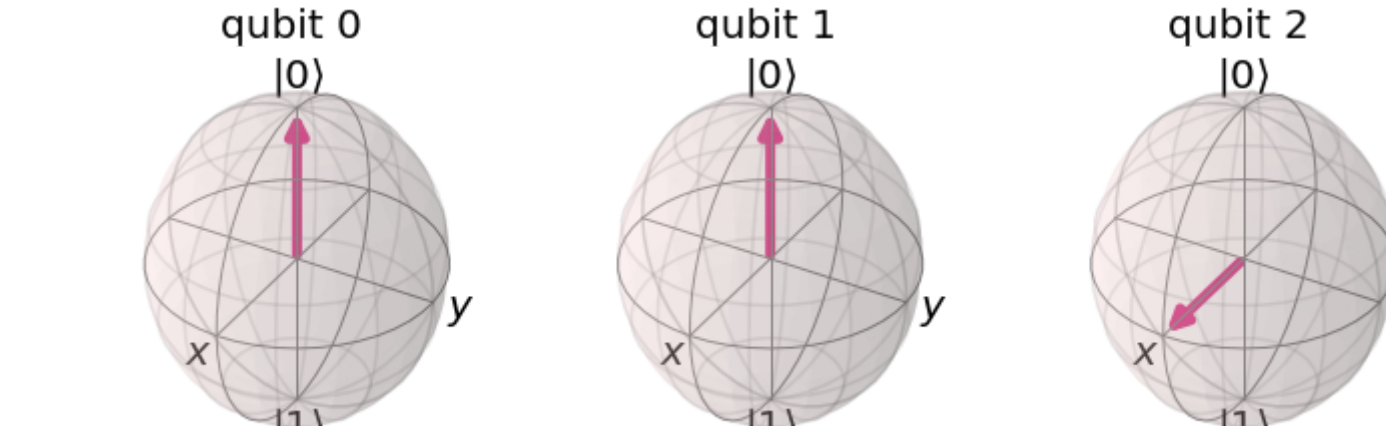
```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



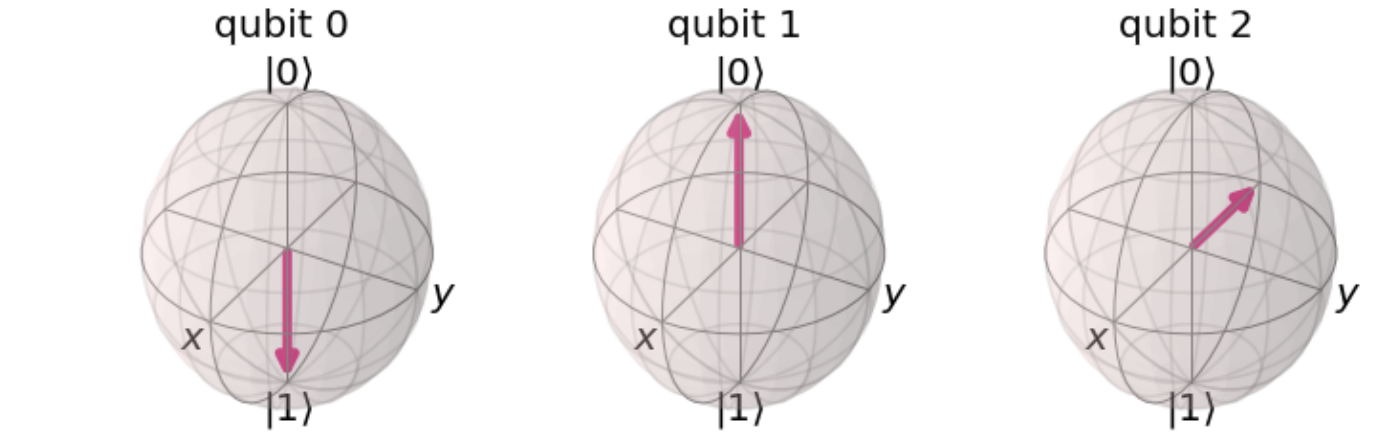
```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



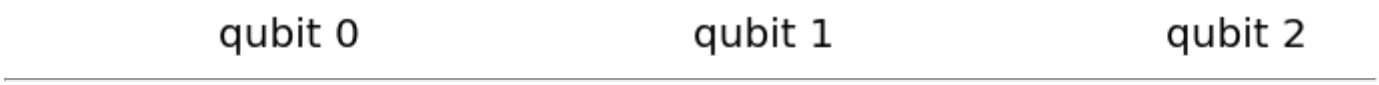
```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```




```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



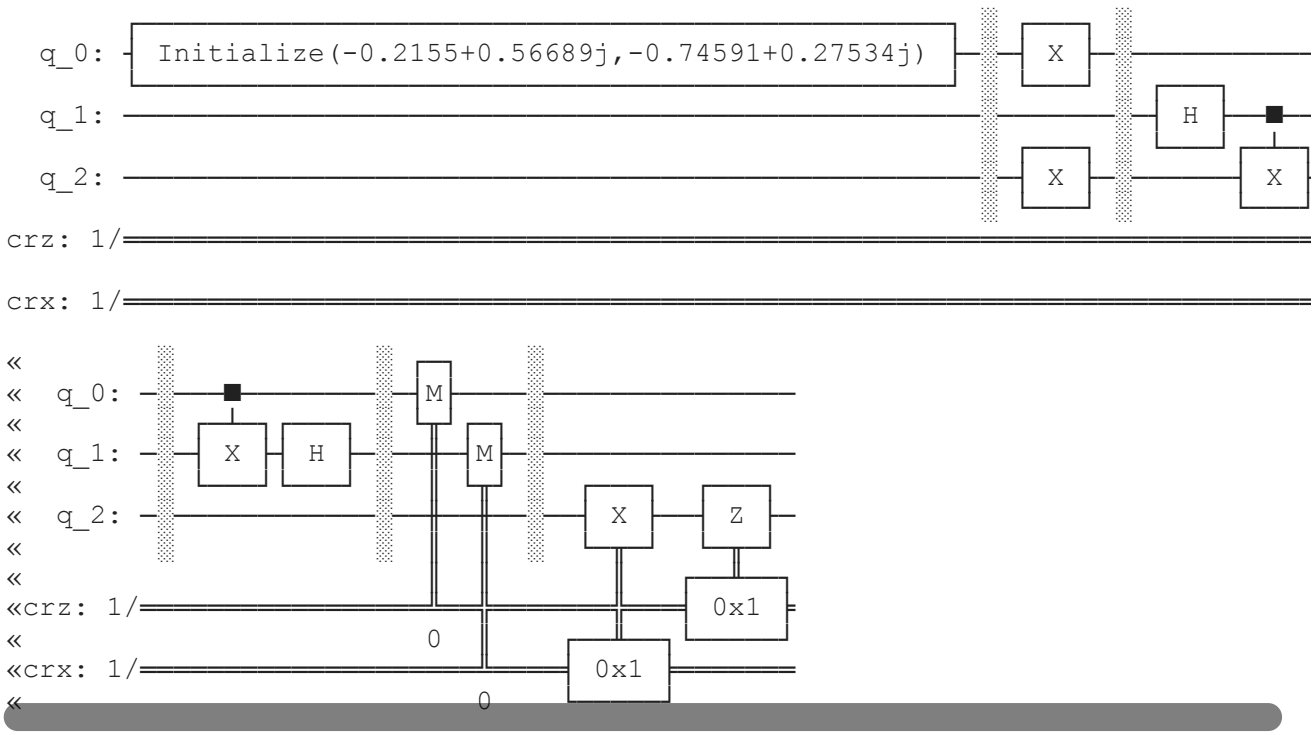

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



3.- TELEPORTING QUANTUM STATES USING DIFFERENT BELL PAIRS

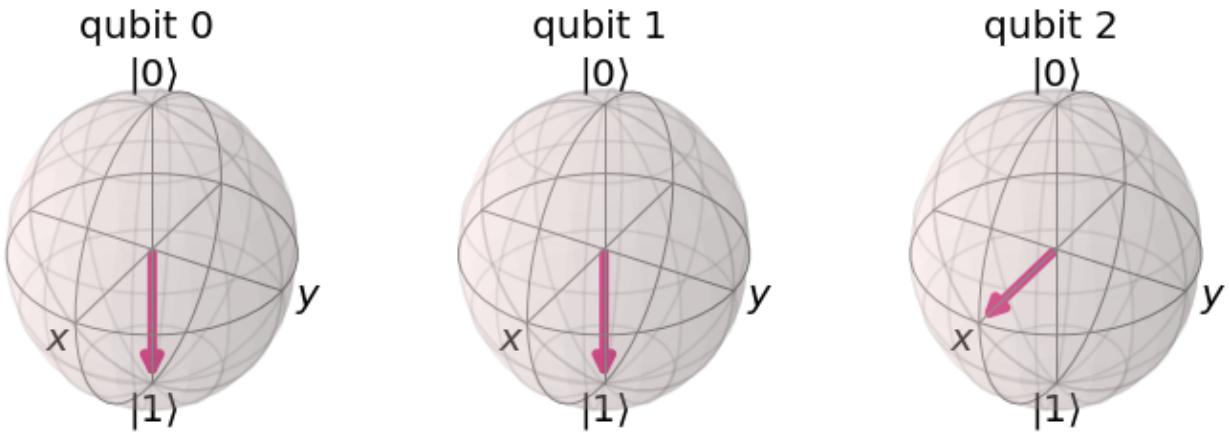
```
def init_state(qc, a):
    #COGEREMOS EL ESTADO RANDOM QUE HEMOS VISTO EN EL CASO D)
    #### your code goes here
    rs = random_state(1)
    qc.initialize(rs,a)
    qc.draw()
    qc.barrier()

    qc.x(0)
    qc.x(2)

    qc.barrier()


def create_bell_pair(qc, a, b):
    """Creates a bell pair in qc using qubits a & b"""
    #### your code goes here
    qc.h(a)
    qc.cx(a,b)
    qc.draw()


def alice_gates(qc, psi, a):
    #### your code goes here
    qc.cx(psi,a)
    qc.h(a)


def measure_and_send(qc, a, b):
    """Measures qubits a & b and 'sends' the results to Bob"""
    qc.barrier()
    #### your code goes here
    qc.measure(a,0)
    qc.measure(b,1)


# This function takes a QuantumCircuit (qc), integer (qubit)
# and ClassicalRegisters (crz & crx) to decide which gates to apply
def bob_gates(qc, qubit, crz, crx):
    #### your code goes here
    qc.x(qubit).c_if(crx,1)
    qc.z(qubit).c_if(crz,1)
```

```
## SETUP APARTADO 3
# Protocol uses 3 qubits and 2 classical bits in 2 different registers
qr = QuantumRegister(3, name="q")
crz, crx = ClassicalRegister(1, name="crz"), ClassicalRegister(1, name="crx")
teleportation_circuit = QuantumCircuit(qr, crz, crx)

## STEP 1
#### your code goes here
init_state(teleportation_circuit,0)



## STEP 2
#### your code goes here
create_bell_pair(teleportation_circuit,1,2)


## STEP 3 & 4
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
alice_gates(teleportation_circuit,0,1)
measure_and_send(teleportation_circuit,0,1)


## STEP 5
teleportation_circuit.barrier() # Use barrier to separate steps
#### your code goes here
bob_gates(teleportation_circuit,2,crz,crx)


teleportation_circuit.draw()
```

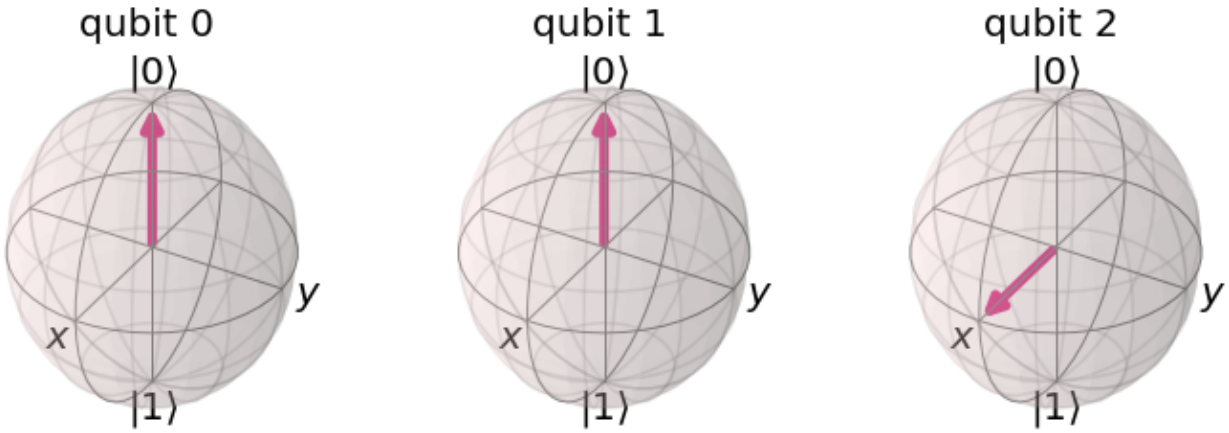



```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
```

```
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

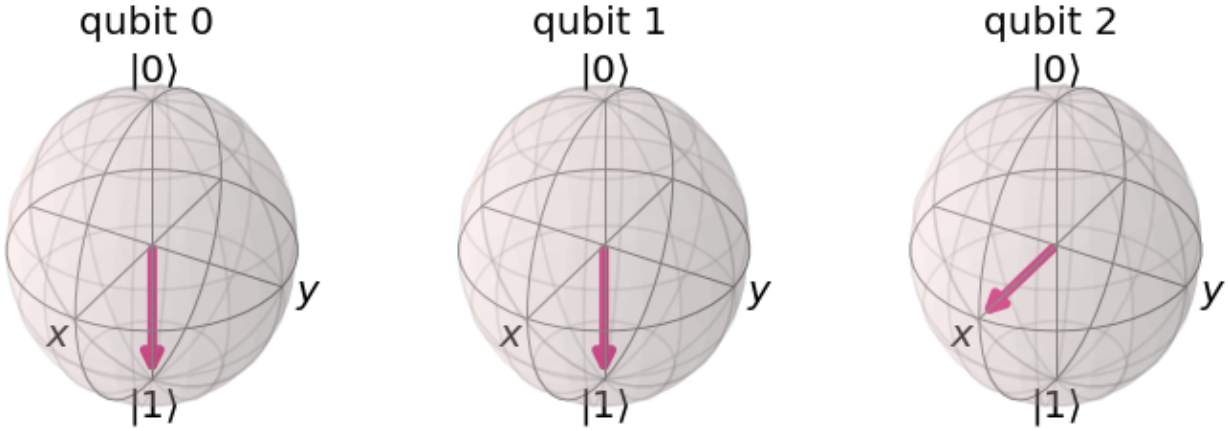


```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

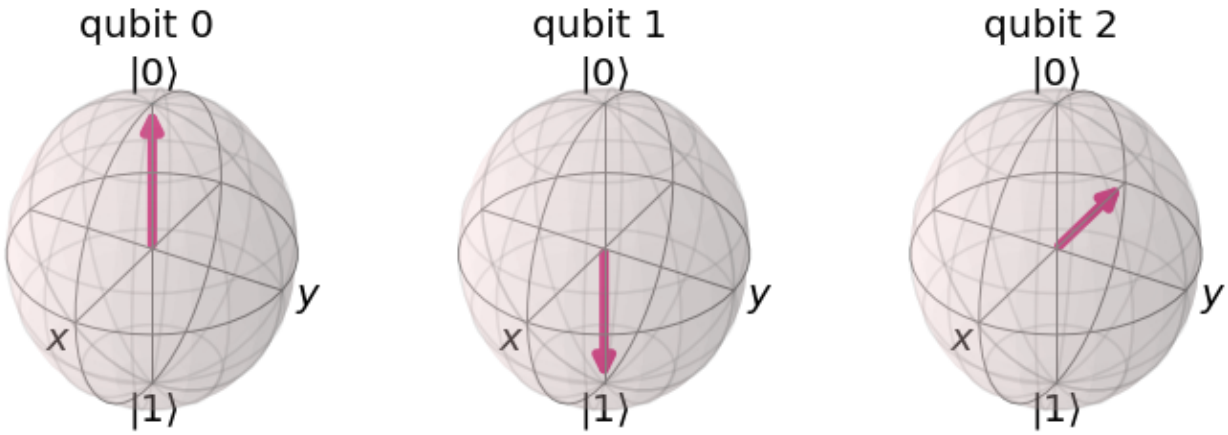


```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```

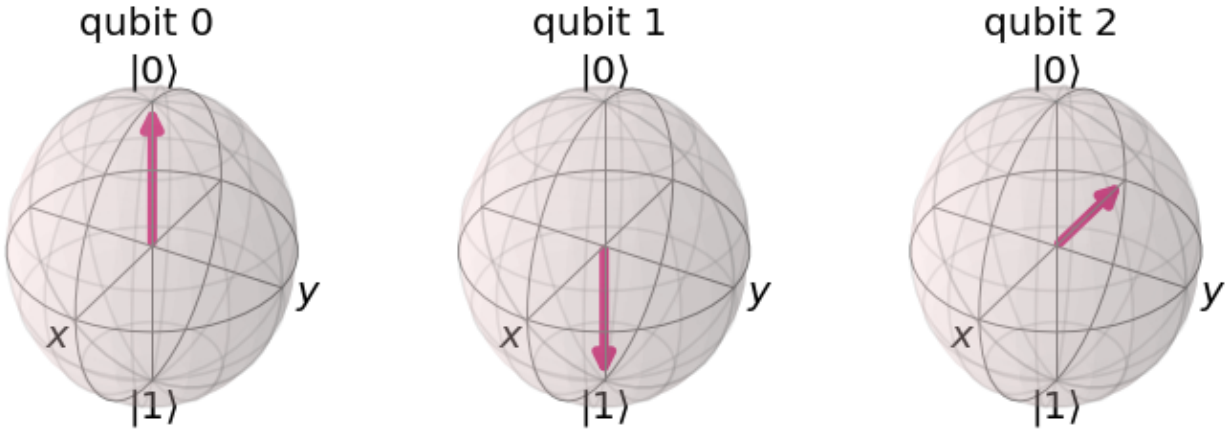

```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```




```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



```
sv_sim = Aer.get_backend('statevector_simulator')
qobj = assemble(teleportation_circuit)
out_vector = sv_sim.run(qobj).result().get_statevector()
plot_bloch_multivector(out_vector)
```



POR CUESTIONES DE TIEMPO Y DESCONOCIMIENTO NO HE PODIDO COMPLETAR LA SEGUNDA PARTE DEL APARTADO 3. NO ESTOY SEGURO DE SI HE REALIZADO CORRECTAMENTE EL RESTO DEL APARTADO. ME GUSTARÍA SABER CÓMO SE REALIZA DE CARA AL EXAMEN. NO PUDE ACUDIR A LA CLASE DE TEORÍA DONDE SE EXPLICÓ ESTO PORQUE ESTABA CONSTIPADO Y PODRÍA SER POSIBLE POSITIVO EN COVID. AL FINAL NO FUE NADA MÁS QUE UN SUSTO. TAMPOCO PUDE ACUDIR A LA SESIÓN DE PRÁCTICAS DE ESA SEMANA PORQUE EL DÍA DE ANTES OPERABAN A MI MADRE. ESPERO QUE LO ENTIENDA. ME GUSTARÍA QUE, SI ES POSIBLE, EN UN CORREO, UNA BREVE EXPLICACIÓN SOBRE CÓMO SE DEBERÍA ACABAR LA PRÁCTICA.

UN SALUDO Y QUE PASE BUEN FIN DE SEMANA,

MIGUEL ÁNGEL.

✓ 0 s completado a las 23:08