MIGUEL ÁNGEL NAVARRO ARENAS

PRÁCTICA 3.

Midiendo qubits y circuitos cuánticos.

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

from qiskit import *
from qiskit.visualization import plot_histogram
from qiskit import QuantumCircuit, execute, Aer
from math import sqrt, pi
import numpy as np
```

PARTE 1.

Midiendo el estado de un qubit.

1) Calcular las probabilidades de medición de los estados de un solo qubit.

Como hemos estado viendo anteriormente, usando el simulador de vectores de Quiskit podemos ver el estado de un qubit, pero esto no es así en la realidad. Cuando medimos un qubit obtenemos un valor binario (0 o 1). Es decir, no podemos leer el estado de salida en superposición obteniendo los valores de alpha y beta.

Para que entendamos mejor que sucede cuando los qubits son medidos, calcularemos las probabilidades de obtener un resultado en la medición de 0 y 1 en los estados cuánticos que veremos a continuación, sabiendo lo siguiente:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2$$

Calcularemos las probabilidades siguientes:

$$1.|\psi_A\rangle = |0\rangle$$
$$2.|\psi_B\rangle = |1\rangle$$
$$3.|\psi_C\rangle = \tfrac{1}{2}|0\rangle + \tfrac{\sqrt{3}}{2}|1\rangle$$
$$4.|\psi_D\rangle = \tfrac{1}{2}|0\rangle + \tfrac{i\sqrt{3}}{2}|1\rangle$$

CÁLCULOS:

ACTIVIDAD 1

$$p(|x\rangle) = |\langle x|\psi\rangle|^2$$

1. $|\psi_A\rangle = |0\rangle$

   $p(|0\rangle) = |\langle 0|\psi_A\rangle|^2$ → $|\langle 0|0\rangle|^2$, $|\langle 0|1\rangle|^2$

   - $|\langle 0|0\rangle|^2 = 1^2 = \boxed{1}$
   - $|\langle 1|0\rangle|^2 = 0^2 = \boxed{0}$

2. $|\psi_B\rangle = |1\rangle$

   $p(|1\rangle) = |\langle 1|\psi_B\rangle|^2$ → $|\langle 1|0\rangle|^2$, $|\langle 1|1\rangle|^2$

   - $|\langle 1|0\rangle|^2 = 0^2 = \boxed{0}$
   - $|\langle 1|1\rangle|^2 = 1^2 = \boxed{1}$

3. $|\psi_C\rangle = \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$

   - $|\langle 0|\psi_C\rangle|^2 = |\frac{1}{2}\langle 0|0\rangle + \frac{\sqrt{3}}{2}\langle 0|1\rangle|^2 = (\frac{1}{2}\cdot 1 + \frac{\sqrt{3}}{2}\cdot 0)^2 = (\frac{1}{2})^2 = \boxed{\frac{1}{4}}$
   - $|\langle 1|\psi_C\rangle|^2 = |\frac{1}{2}\langle 1|0\rangle + \frac{\sqrt{3}}{2}\langle 1|1\rangle|^2 = (\frac{1}{2}\cdot 0 + \frac{\sqrt{3}}{2}\cdot 1)^2 = \boxed{\frac{3}{4}}$

4. $|\psi_D\rangle = \frac{1}{2}\cdot|0\rangle + \frac{i\cdot\sqrt{3}}{2}|1\rangle$

   - $|\langle 0|\psi_D\rangle|^2 = |\frac{1}{2}\cdot\langle 0|0\rangle + \frac{i\cdot\sqrt{3}}{2}\langle 0|1\rangle|^2 = |\frac{1}{2}\cdot 1|^2 = \boxed{\frac{1}{4}}$ ($\langle 0|1\rangle = \emptyset$)
   - $+|\langle 1|\psi_D\rangle|^2 = |\frac{1}{2}\cdot\langle 1|0\rangle + \frac{i\cdot\sqrt{3}}{2}\langle 1|1\rangle = |\frac{i\cdot\sqrt{3}}{2}|^2 =$

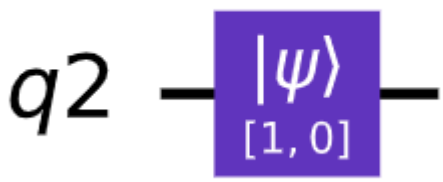
   $= \boxed{\frac{3}{4}}$

¿Son las mediciones de las probabilidades para los apartados C y D iguales? Si lo son, explica por qué los estados cuánticos son diferentes.

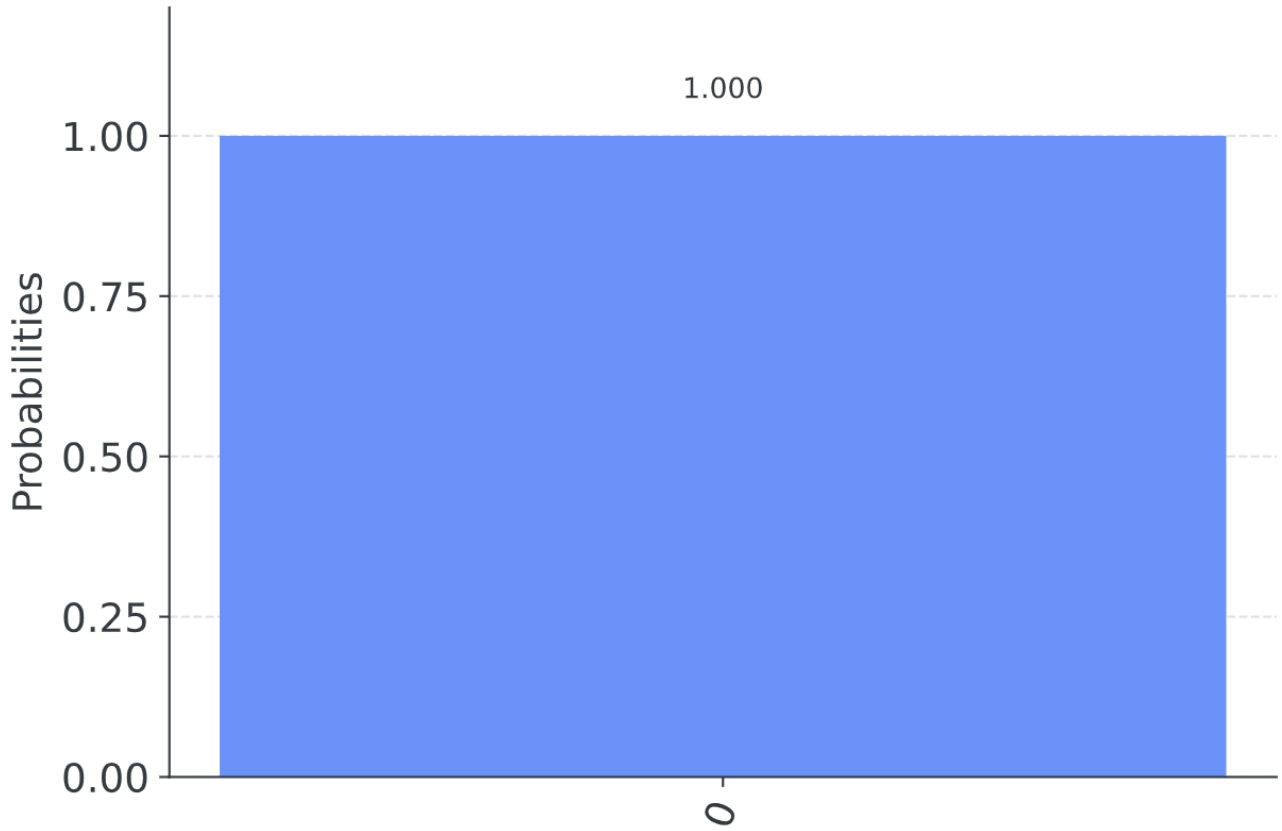
1.1) Midiendo un solo qubit con Qiskit.

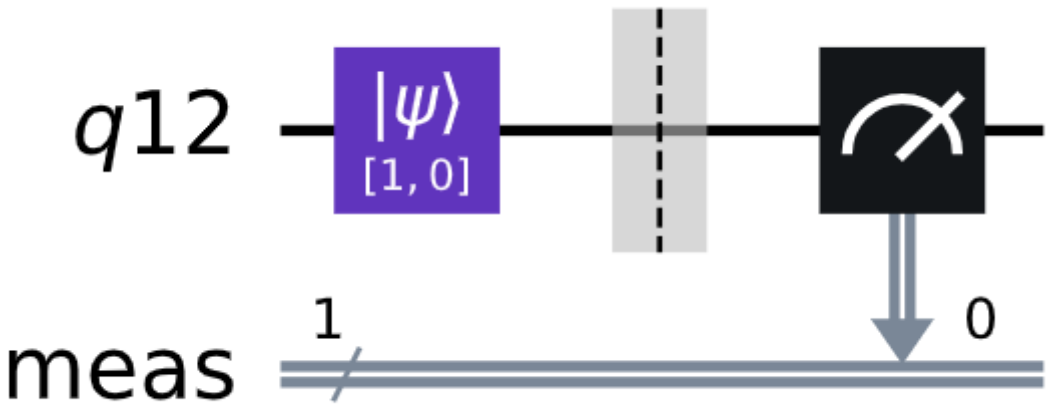
Ahora, verificaremos la medición de las probabilidades obtenidas usando Qiskit. Para ello, tendremos que inicializar un qubit a cada uno de los estados quánticos, en este caso, a 0 y a 1. Debes usar la función initialize.

```
#EJEMPLO DEL SUMMARY (ARBITRARY INITIALIZATION)
import math

#### your code goes here

desired_vector = [
    1 / math.sqrt(16) * complex(0, 1),
    1 / math.sqrt(8) * complex(1, 0),
    1 / math.sqrt(16) * complex(1, 1),
    0,
    0,
    1 / math.sqrt(8) * complex(1, 2),
    1 / math.sqrt(16) * complex(1, 0),
    0]
# Let's view our circuit

q = QuantumRegister(3)

qc = QuantumCircuit(q)

qc.initialize(desired_vector, [q[0],q[1],q[2]])

qc.draw()


#CASO 1.
desired_vector=[1,0]
q=QuantumRegister(1)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0]])
qc.draw()
```

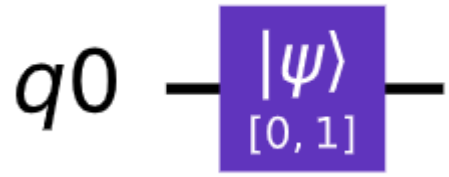


```
Statevector([1.+0.j, 0.+0.j],
            dims=(2,))
```

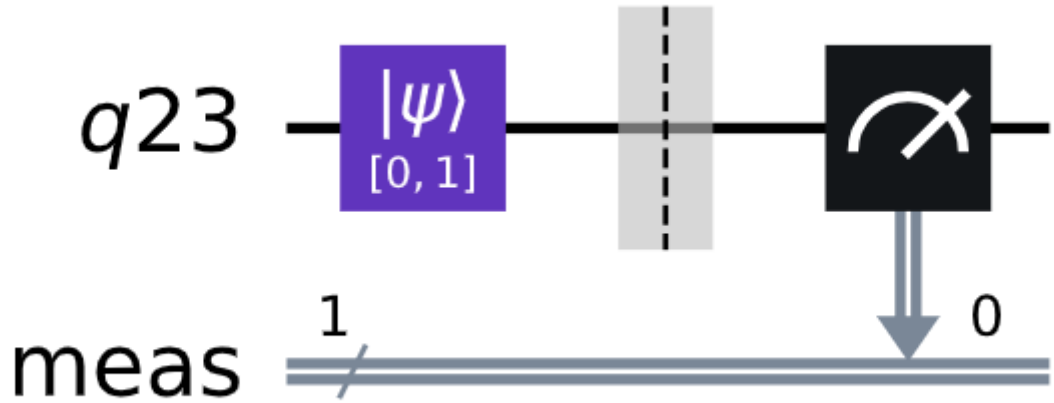


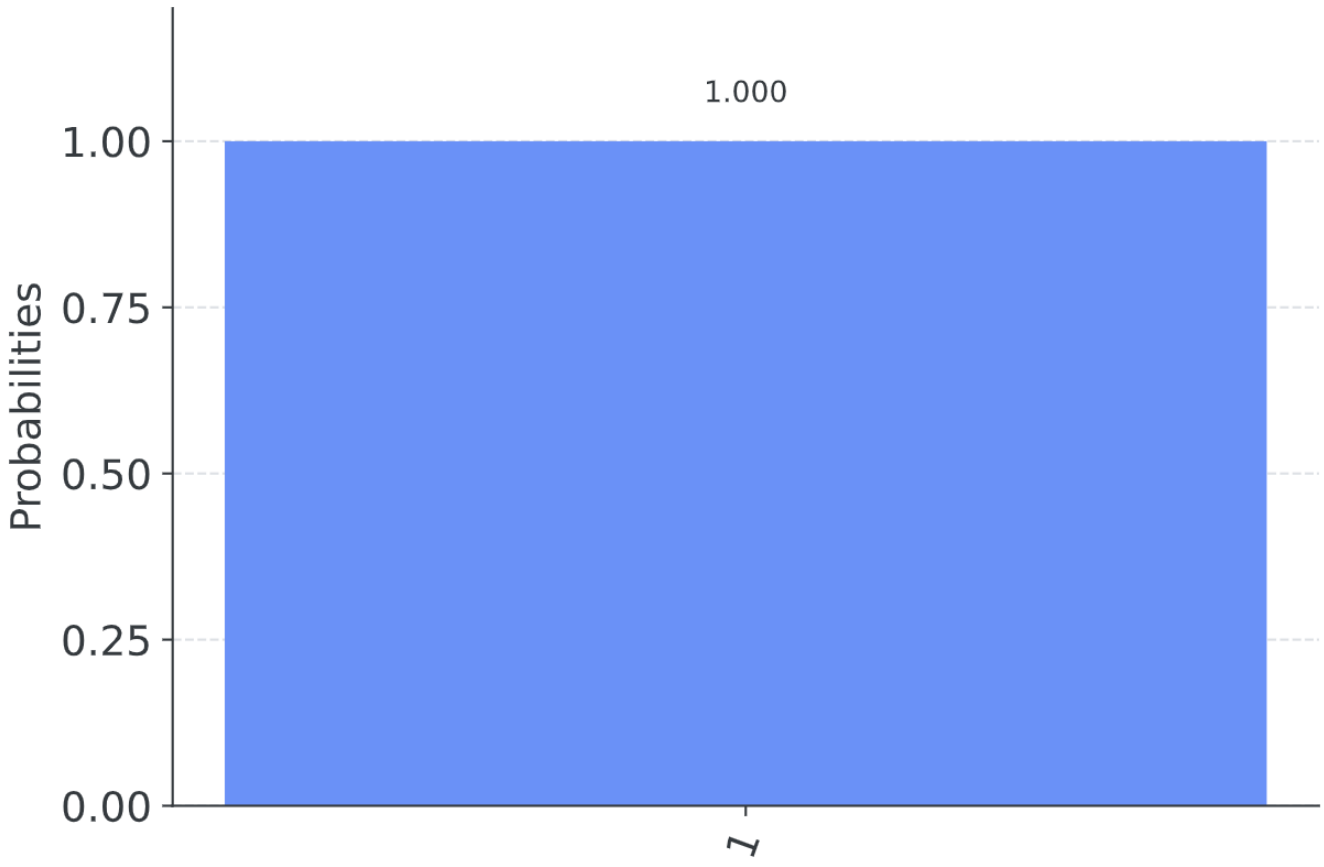



```
#CASO 2.
desired_vector=[0,1]
q=QuantumRegister(1)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0]])
qc.draw()
```



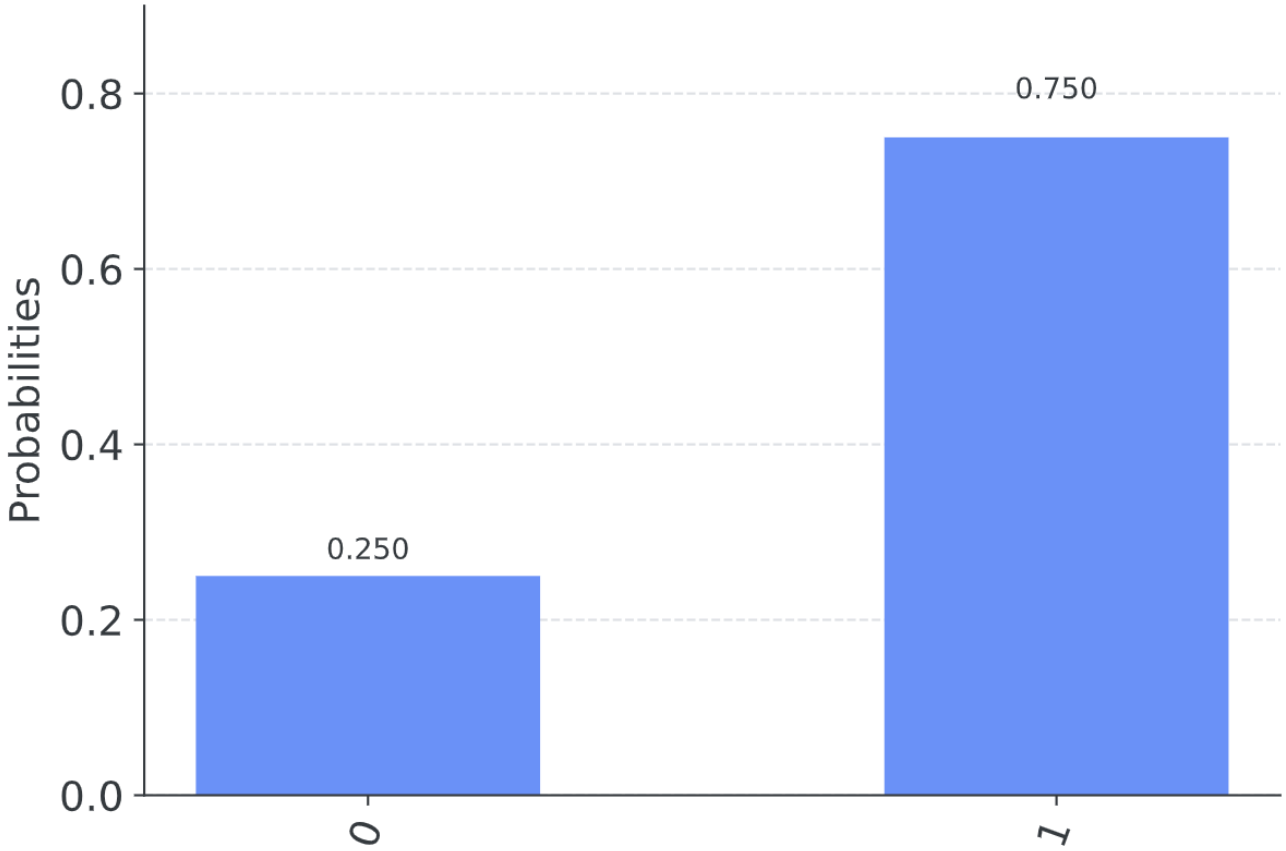

```
Statevector([0.+0.j, 1.+0.j],
            dims=(2,))
```

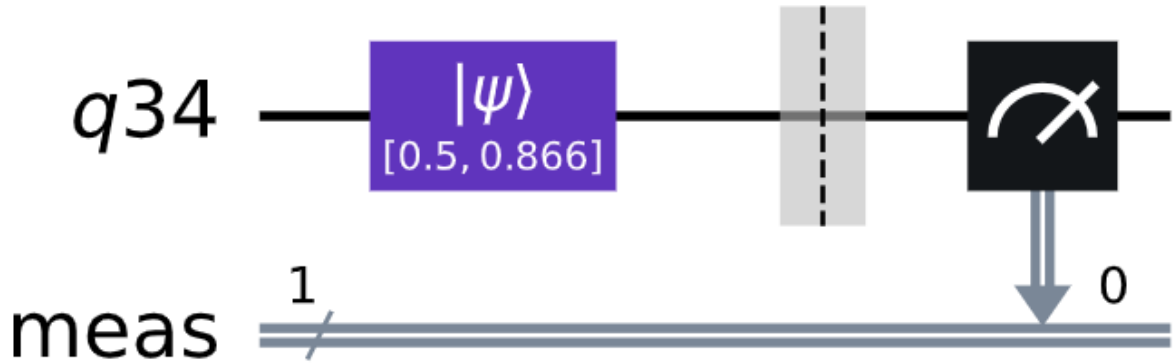




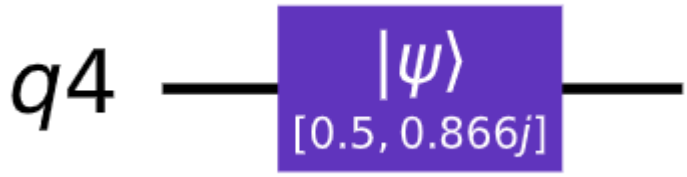
```
#CASO 3.
import math
desired_vector=[1/2,math.sqrt(3)/2]
q=QuantumRegister(1)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0]])
qc.draw()
```


```
Statevector([0.5       +0.j, 0.8660254+0.j],
            dims=(2,))
```

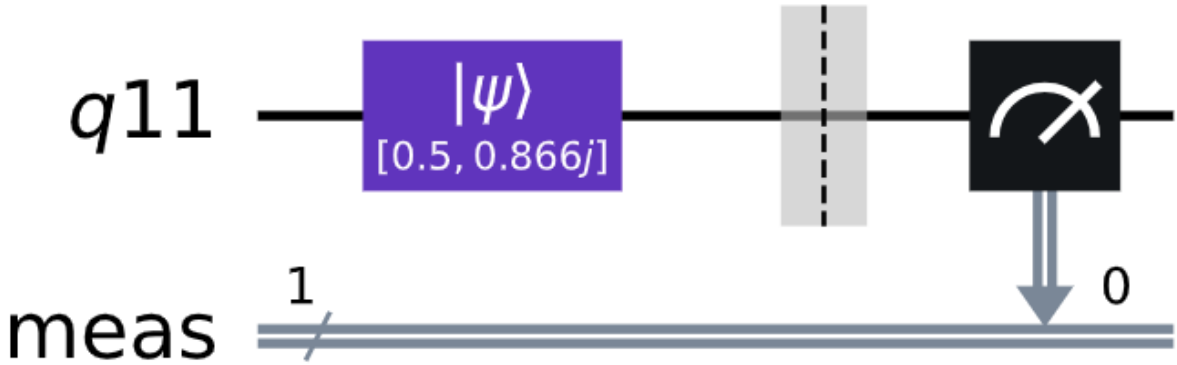


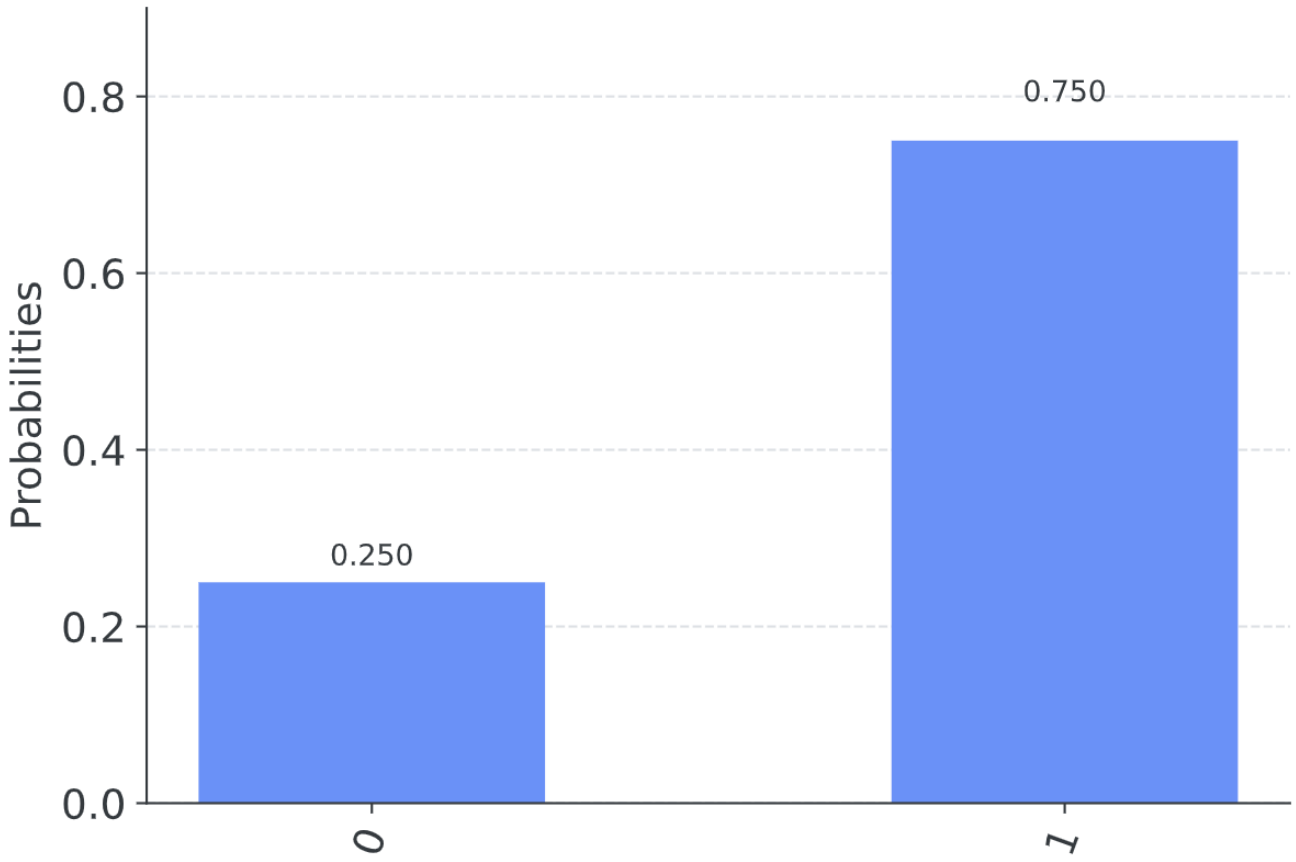



```
#CASO 4.
import math
desired_vector=[(1/2),(complex(0,math.sqrt(3))/2)]
q=QuantumRegister(1)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0]])
qc.draw()
```



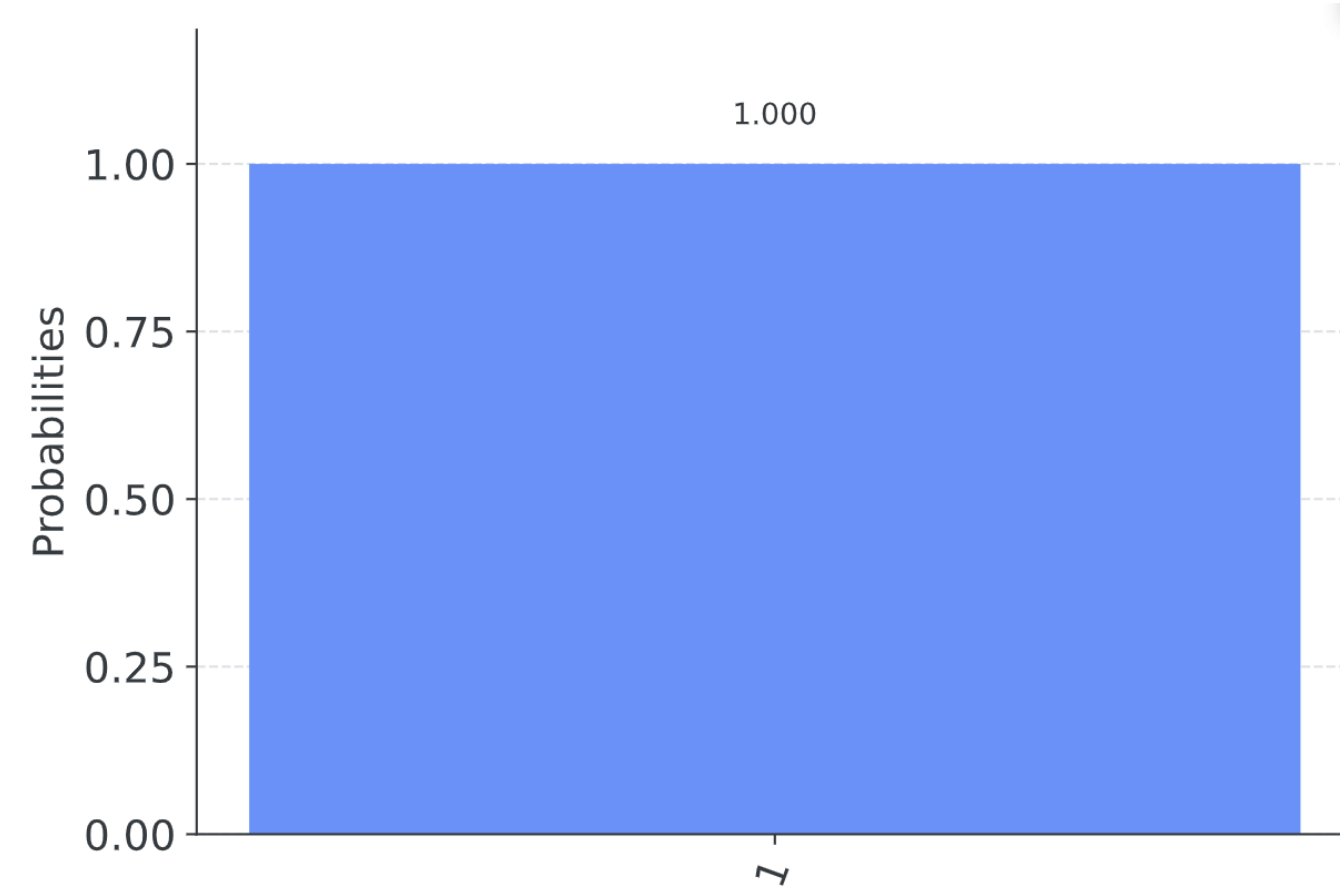

```
Statevector([0.5+0.j       , 0. +0.8660254j],
            dims=(2,))
```

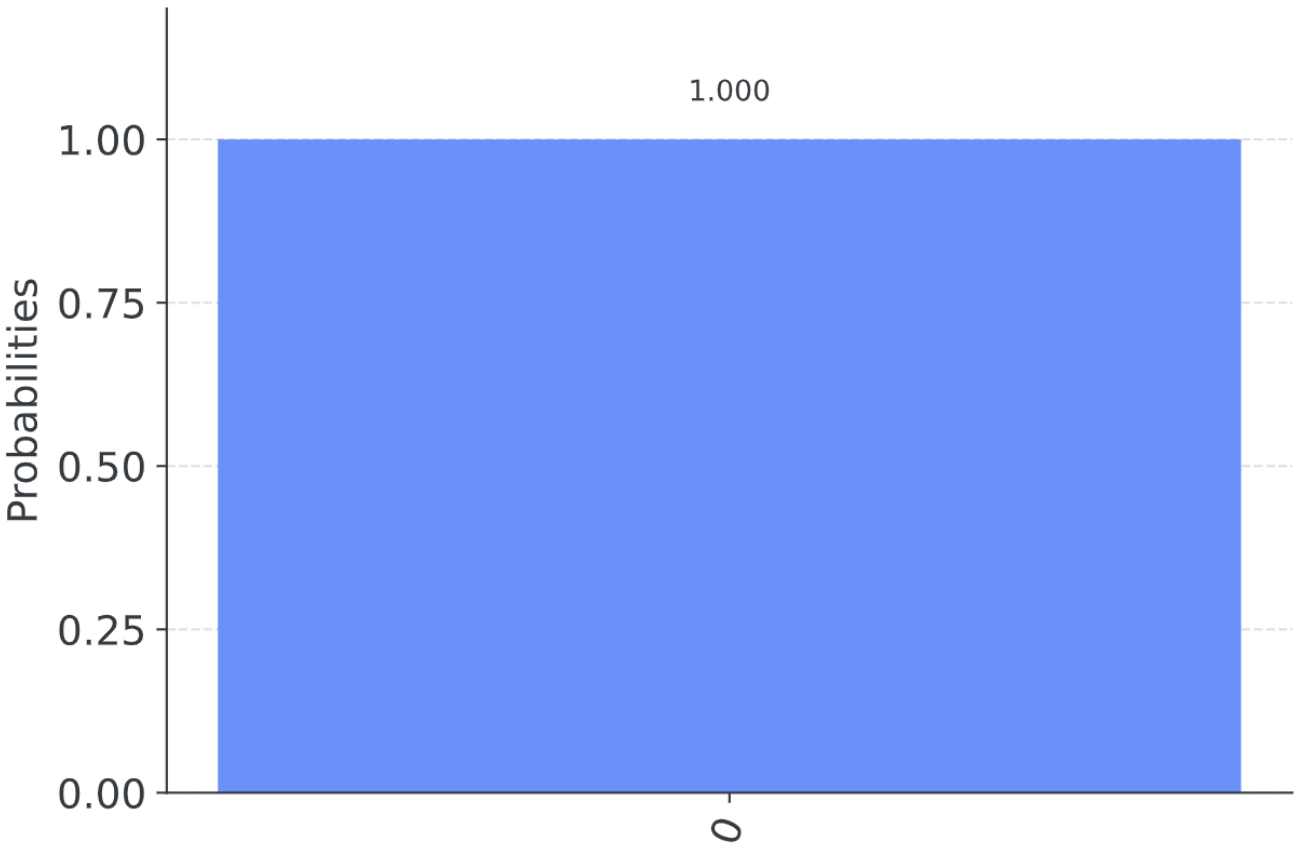




EJERCICIO: ejecuta el código siguiente varias veces, al menos diez, y mira lo que observas:

En este caso, vemos que algunas veces obtenemos un 0 y otras veces obtenemos un 1, siendo mucho mayor la probabilidad de que nos aparezca un 1 (0.75 del 1 vs 0.25 del 0). Hemos tomado el caso 4 como ejemplo.





1.2.- Midiendo un qubit varias veces con quiskit

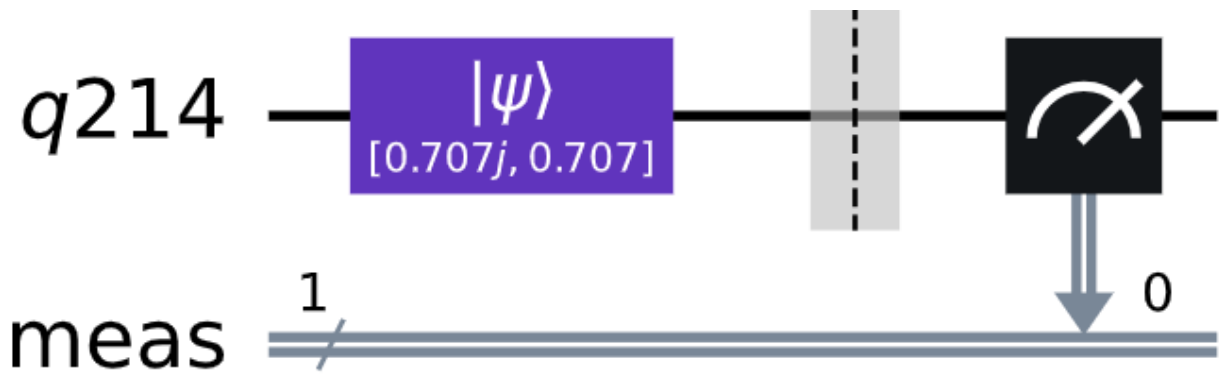
```
qc = QuantumCircuit(1)  # Create a quantum circuit with one qubit
#### your code goes here
import math
desired_vector = [
    1 / math.sqrt(2) * complex(0, 1),
    1 / math.sqrt(2) * complex(1, 0),
]
```

```
q = QuantumRegister(1)

qc = QuantumCircuit(q)

qc.initialize(desired_vector, q[0])
qc.draw()
```

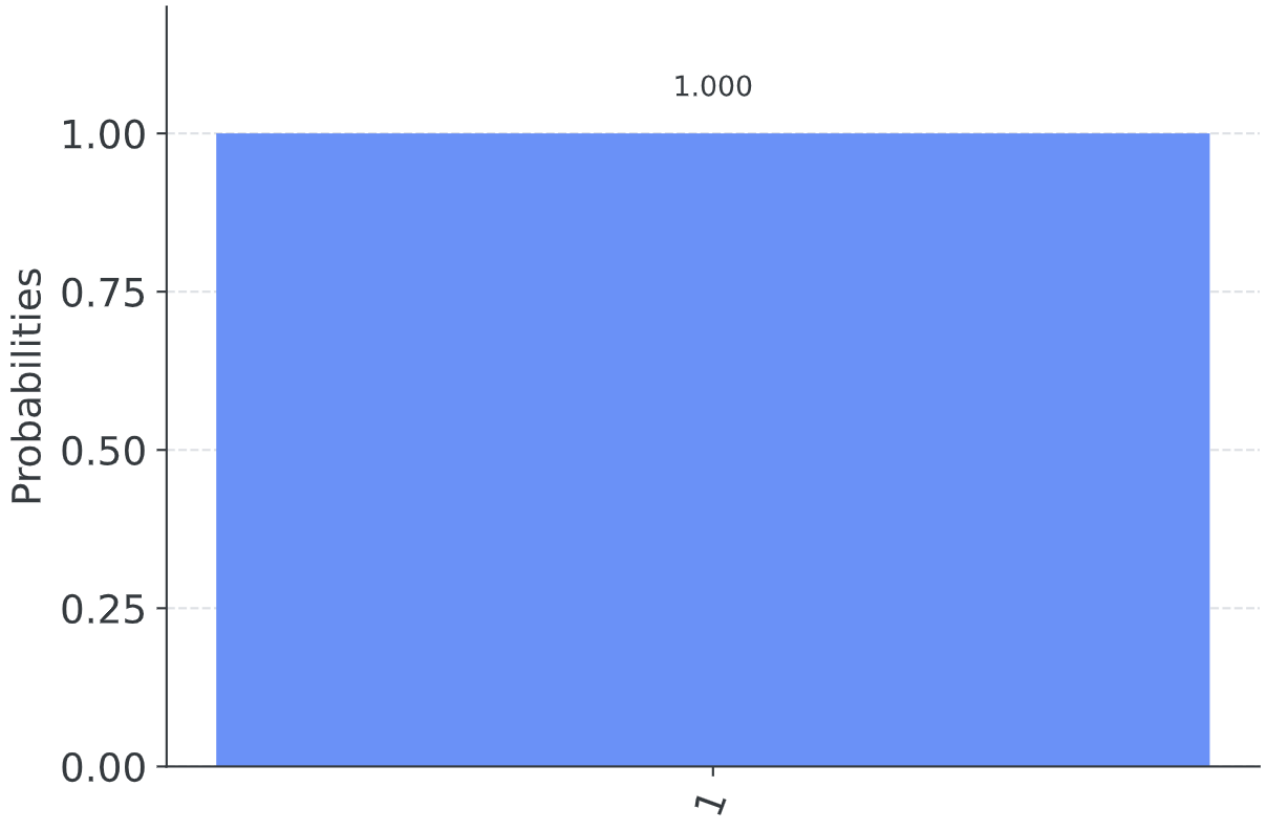
```
# Let's display the output state vector
result = execute(qc,backend).result() # Do the simulation, returning the result
out_state = result.get_statevector()
print(out_state) # Display the output state vector
```

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

```
backend = Aer.get_backend('statevector_simulator') # Tell Qiskit how to simulate ou
qc.measure_all()
qc.draw()
```



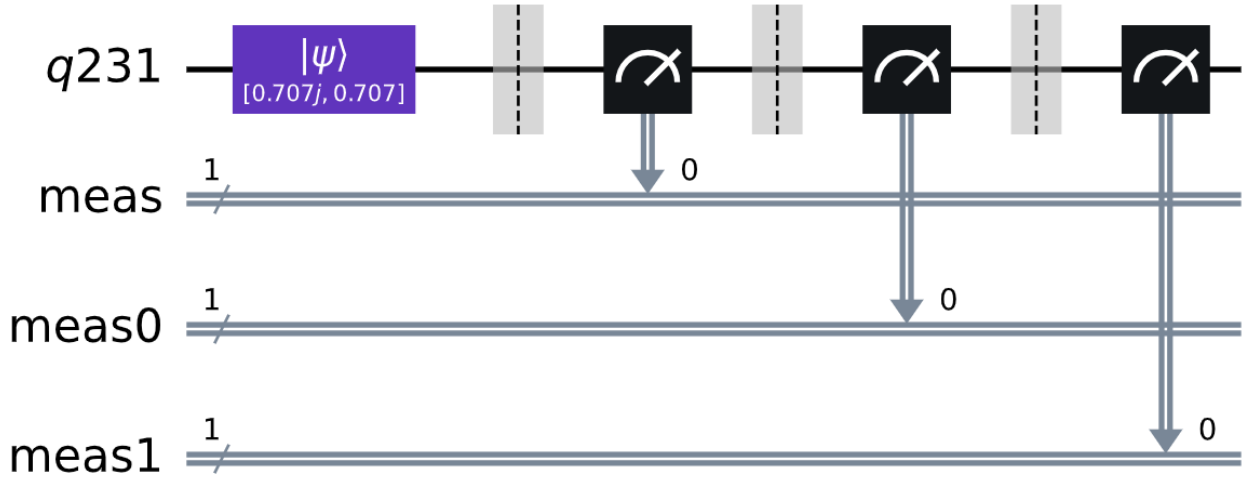

```
results = execute(qc,backend).result().get_counts()
plot_histogram(results)
```



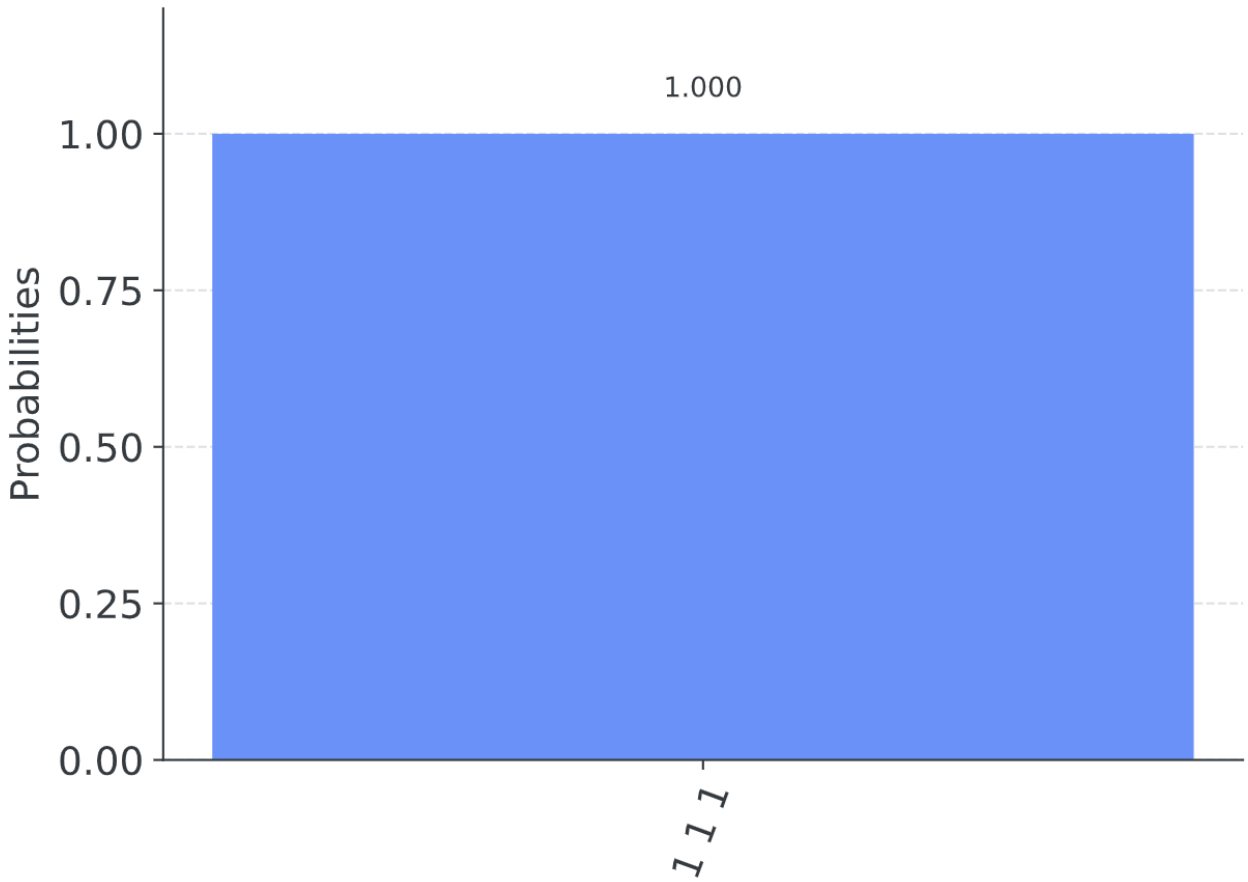

```
qc = QuantumCircuit(1)  # Create a quantum circuit with one qubit
#### your code goes here
import math
desired_vector = [
    1 / math.sqrt(2) * complex(0, 1),
    1 / math.sqrt(2) * complex(1, 0),
]


q = QuantumRegister(1)

qc = QuantumCircuit(q)

qc.initialize(desired_vector, q[0])
qc.draw()



# Let's display the output state vector
result = execute(qc,backend).result() # Do the simulation, returning the result
out_state = result.get_statevector()
print(out_state) # Display the output state vector
```

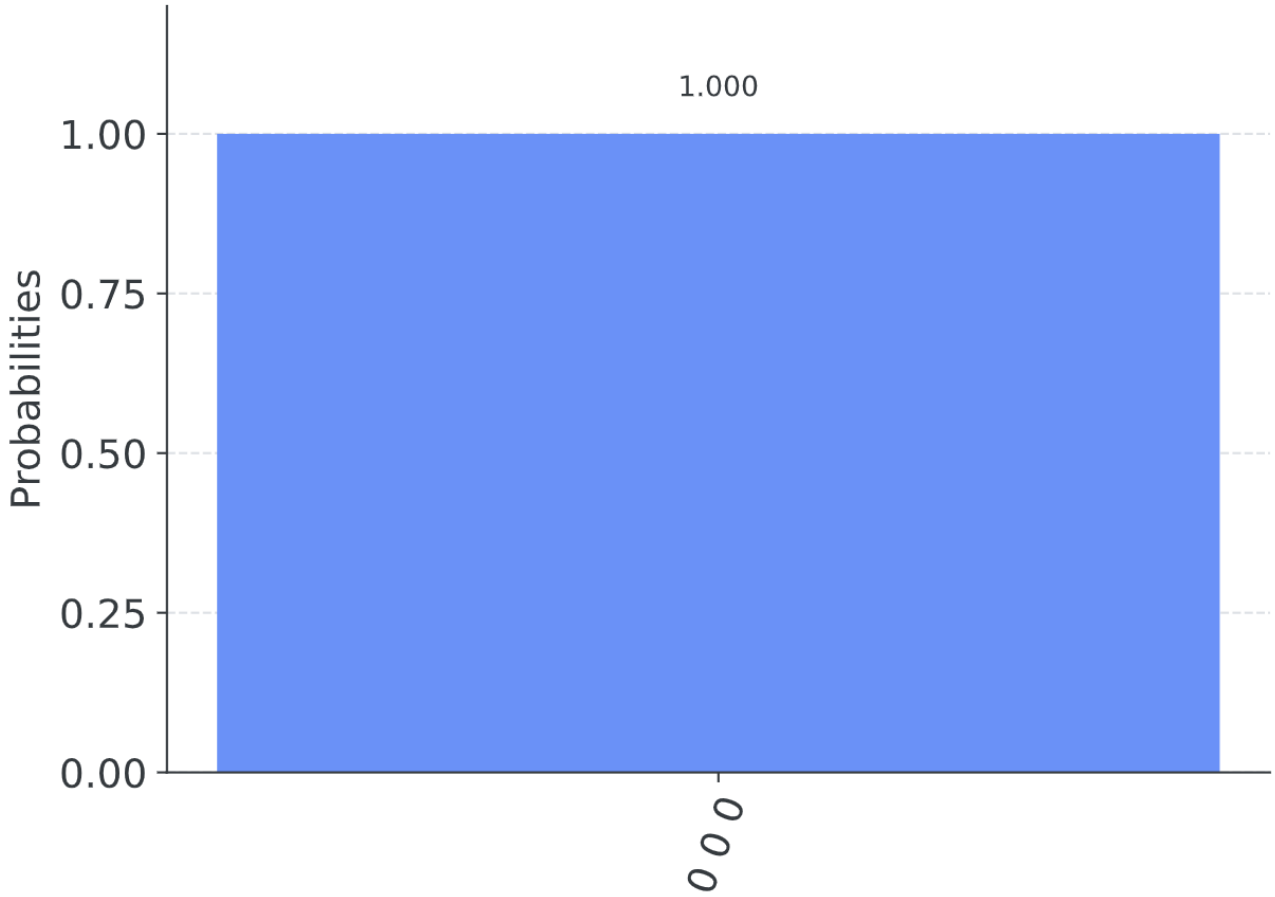
```
Statevector([0.        +0.70710678j, 0.70710678+0.j        ],
            dims=(2,))
```

```
backend = Aer.get_backend('statevector_simulator') # Tell Qiskit how to simulate ou
qc.measure_all()
qc.draw()
```




```
results = execute(qc,backend).result().get_counts()
plot_histogram(results)
```





PORQUE SI LA PRIMERA VEZ QUE MEDIMOS UN CERO O UN 1 LOS DEMÁS RESULTADOS TAMBIÉN VAN A SER EL MISMO?

Porque cuando medimos rompemos el estado cuántico, es decir, descuantificamos. Una vez lo oservamos ya sabemos que si o si va a estar todo el tiempo en ese estado que hemos observado inicialmente todas las veces que lo midamos después.

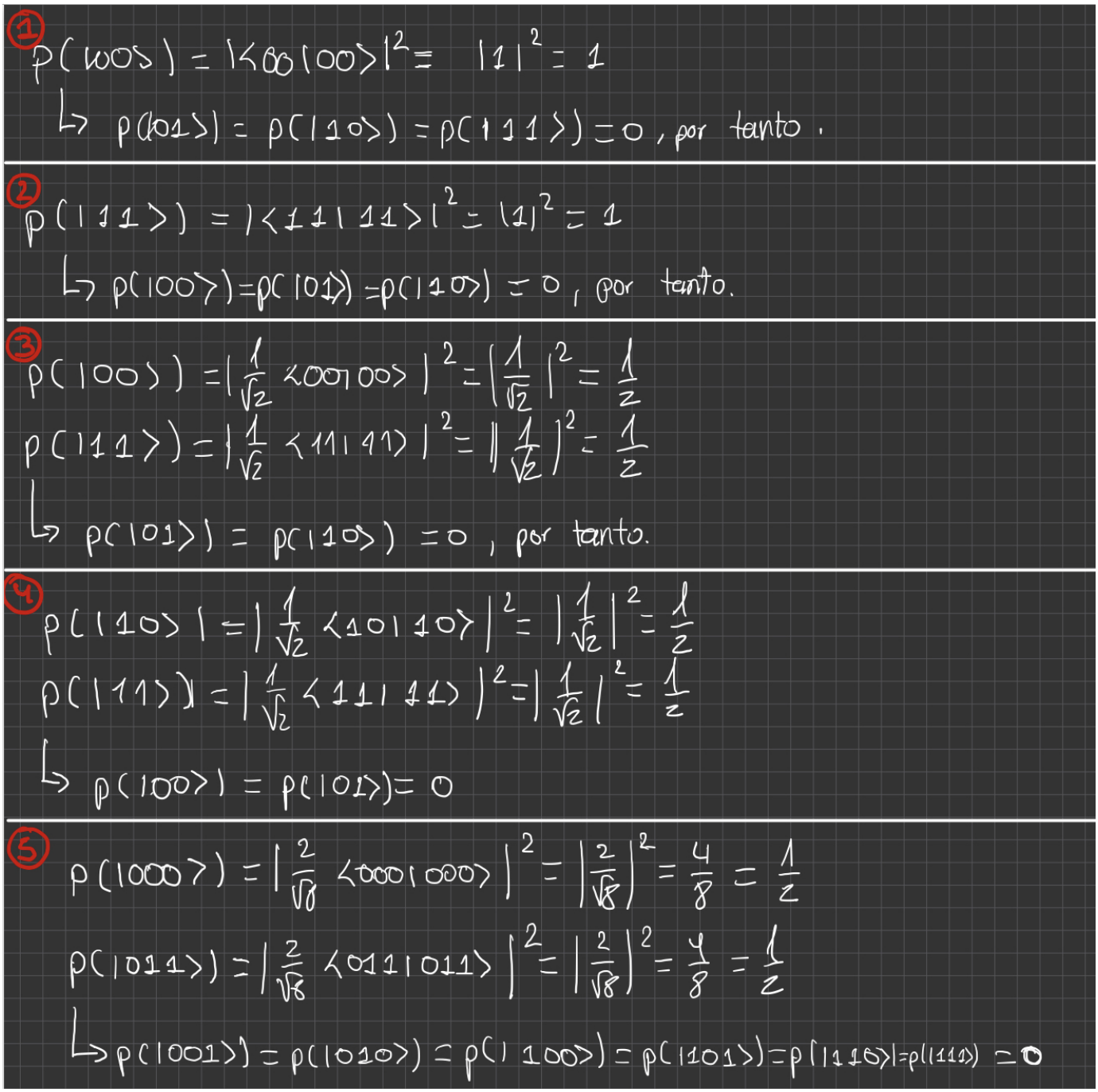
PARTE 2

CALCULAR LAS PROBABILIDADES DE MEDICIÓN DE MULTIPLES ESTADOS DE QUBITS.

ACTIVIDAD

Calcula a mano la probabilidad de los siguientes estados cuánticos:

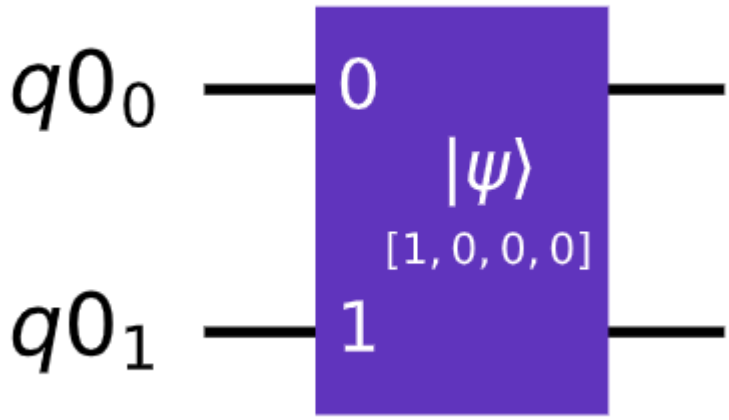
$$1.|\psi_A\rangle = |00\rangle$$
$$2.|\psi_B\rangle = |11\rangle$$
$$3.|\psi_C\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$
$$4.|\psi_D\rangle = \frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$
$$5.|\psi_E\rangle = \frac{2}{\sqrt{8}}|000\rangle + \frac{2}{\sqrt{8}}|011\rangle$$



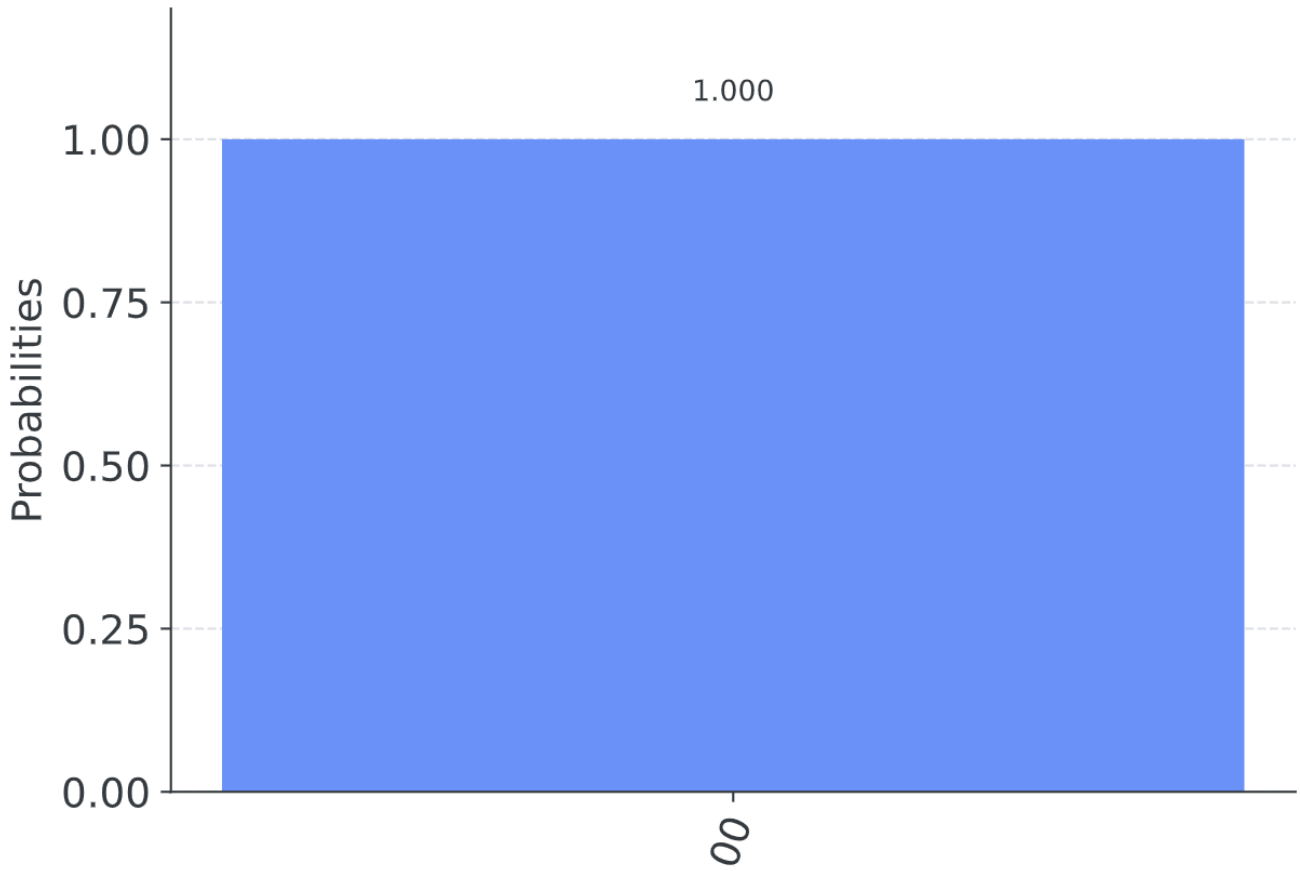
2.1.- CREA EL QUIBIT EN EL ESTADO " $|\psi_C\rangle$ " USANDO UN CIRCUITO

NOTA: en nuestro caso, hemos creado el circuito de todos los estados cuánticos que aparecían en el punto 2 (fotografía anterior) y hemos obtenido los siguientes resultados:

```
#CASO 1.
import math
desired_vector=[1,0,0,0]
q=QuantumRegister(2)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0],q[1]])
qc.draw()
```

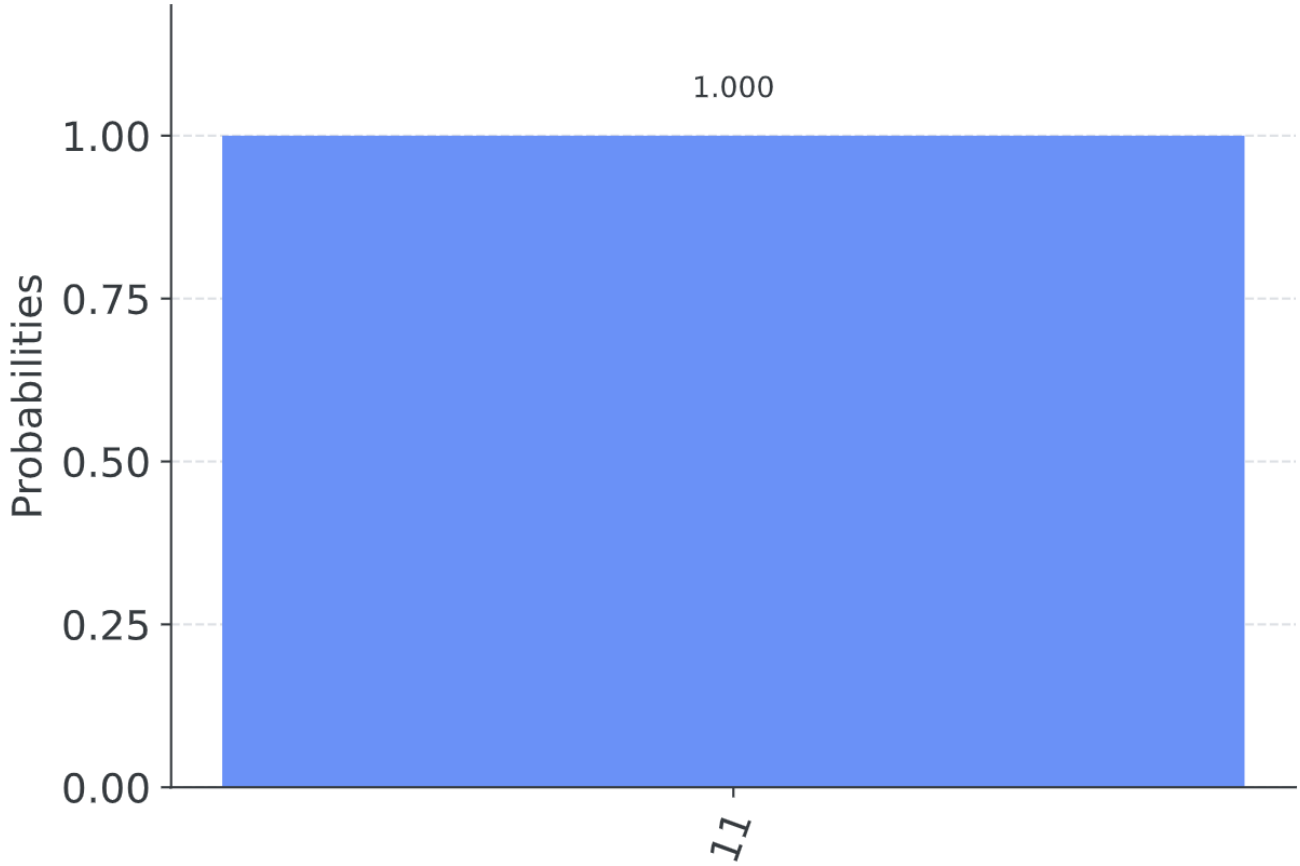



```
Statevector([1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
            dims=(2, 2))
```

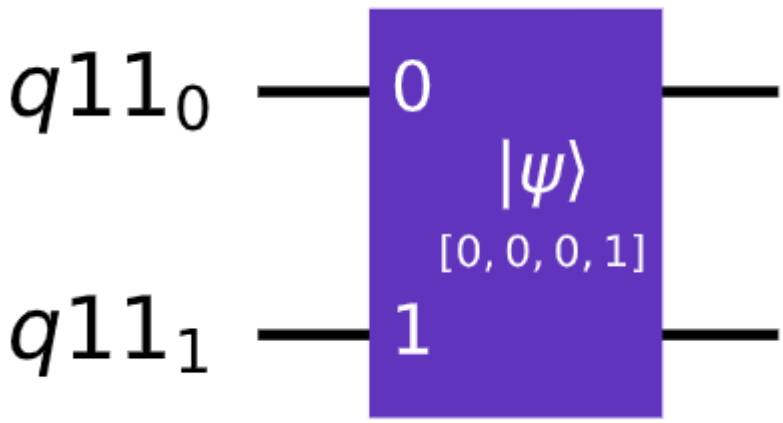



```
#CASO 2.
import math
desired_vector=[0,0,0,1]
q=QuantumRegister(2)
qc=QuantumCircuit(q)
```

```
qc.initialize(desired_vector, [q[0],q[1]])
qc.draw()
```

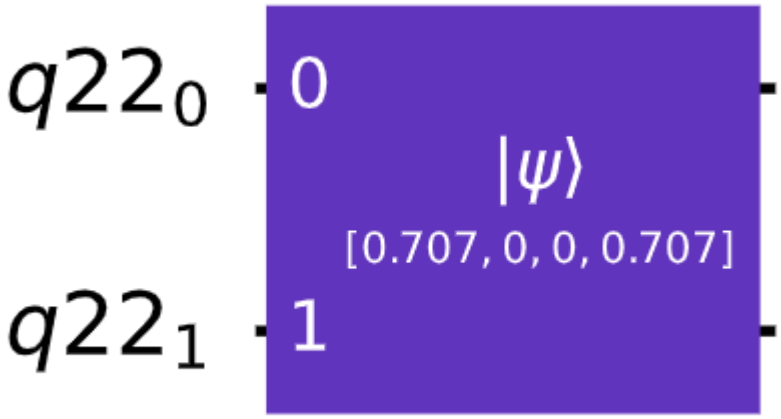


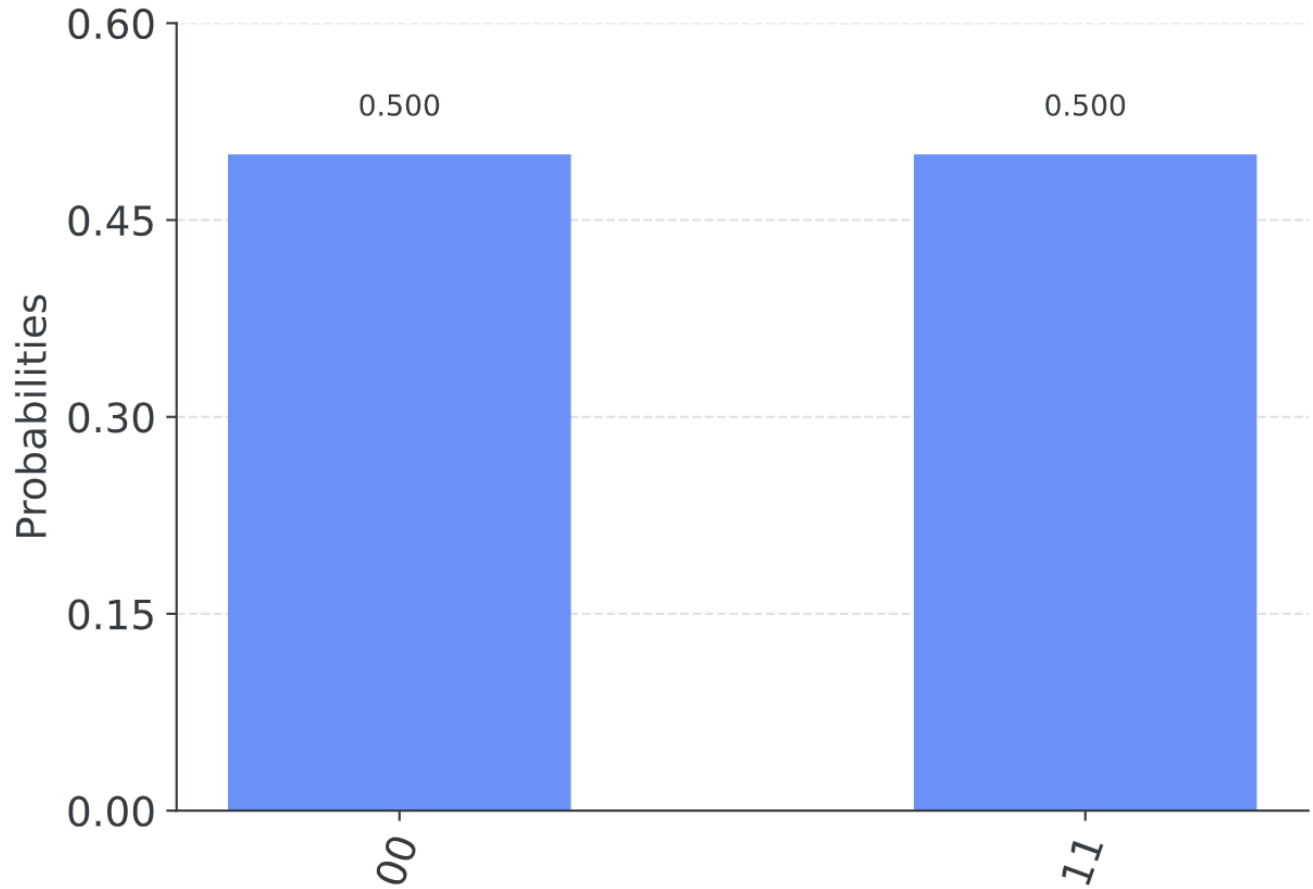

```
Statevector([0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j],
            dims=(2, 2))
```




```
#CASO 3.
import math
desired_vector=[1/math.sqrt(2),0,0,1/math.sqrt(2)]
q=QuantumRegister(2)
qc=QuantumCircuit(q)
```

```
qc.initialize(desired_vector, [q[0],q[1]])
qc.draw()
```

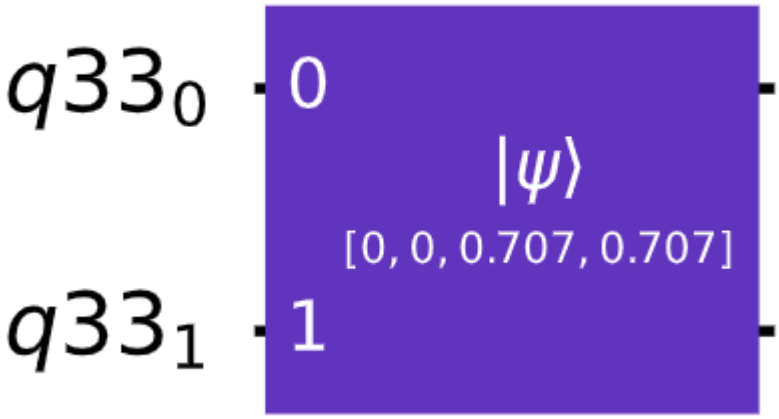


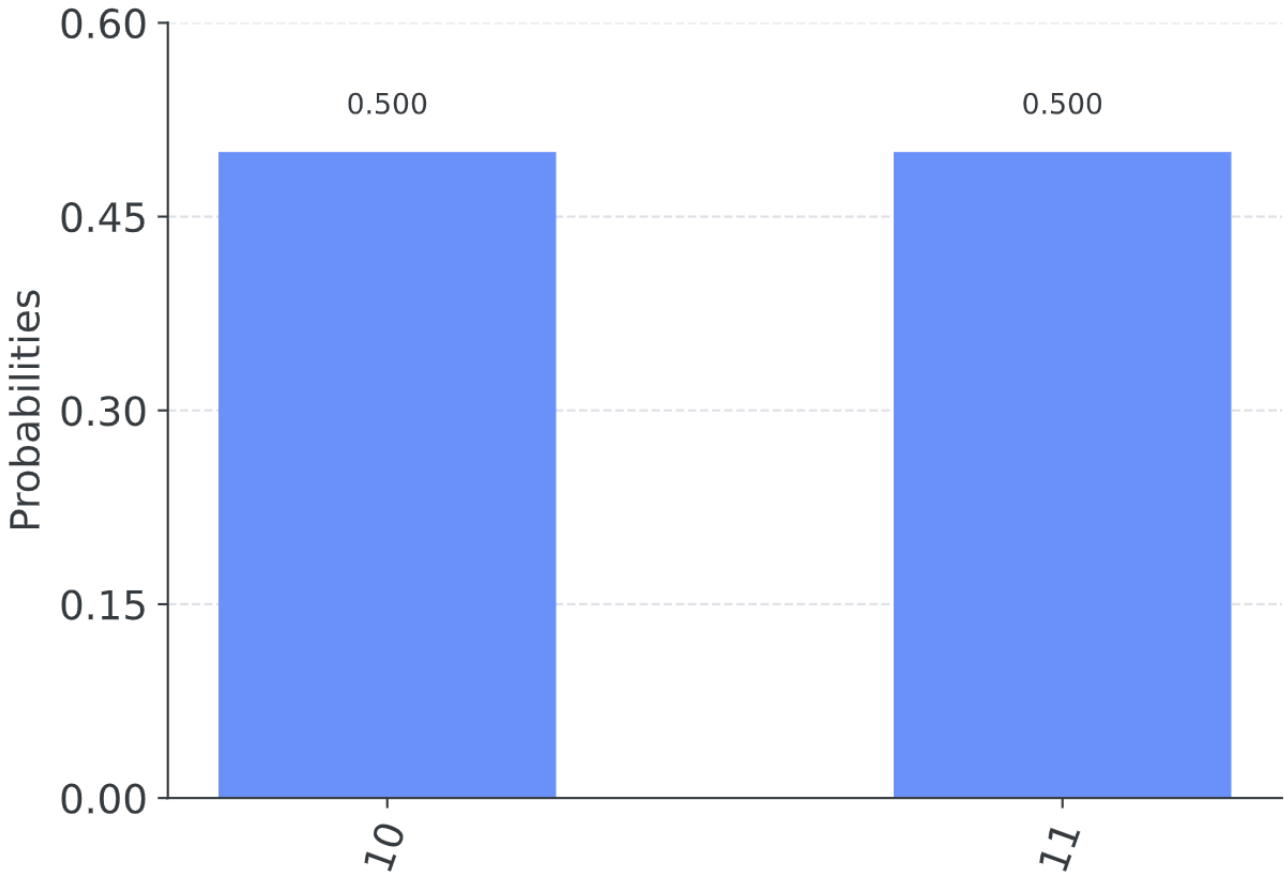



```
Statevector([0.70710678+0.j, 0.        +0.j, 0.        +0.j,
             0.70710678+0.j],
            dims=(2, 2))
```

```
#CASO 4.
```

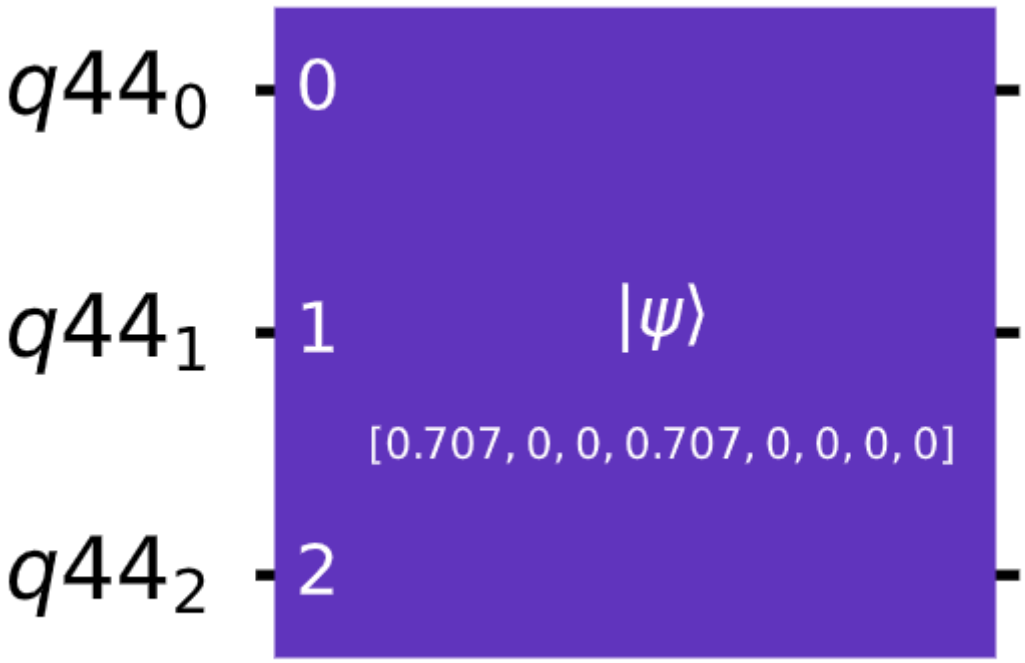
```
import math
desired_vector=[0,0,1/math.sqrt(2),1/math.sqrt(2)]
q=QuantumRegister(2)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0],q[1]])
qc.draw()
```

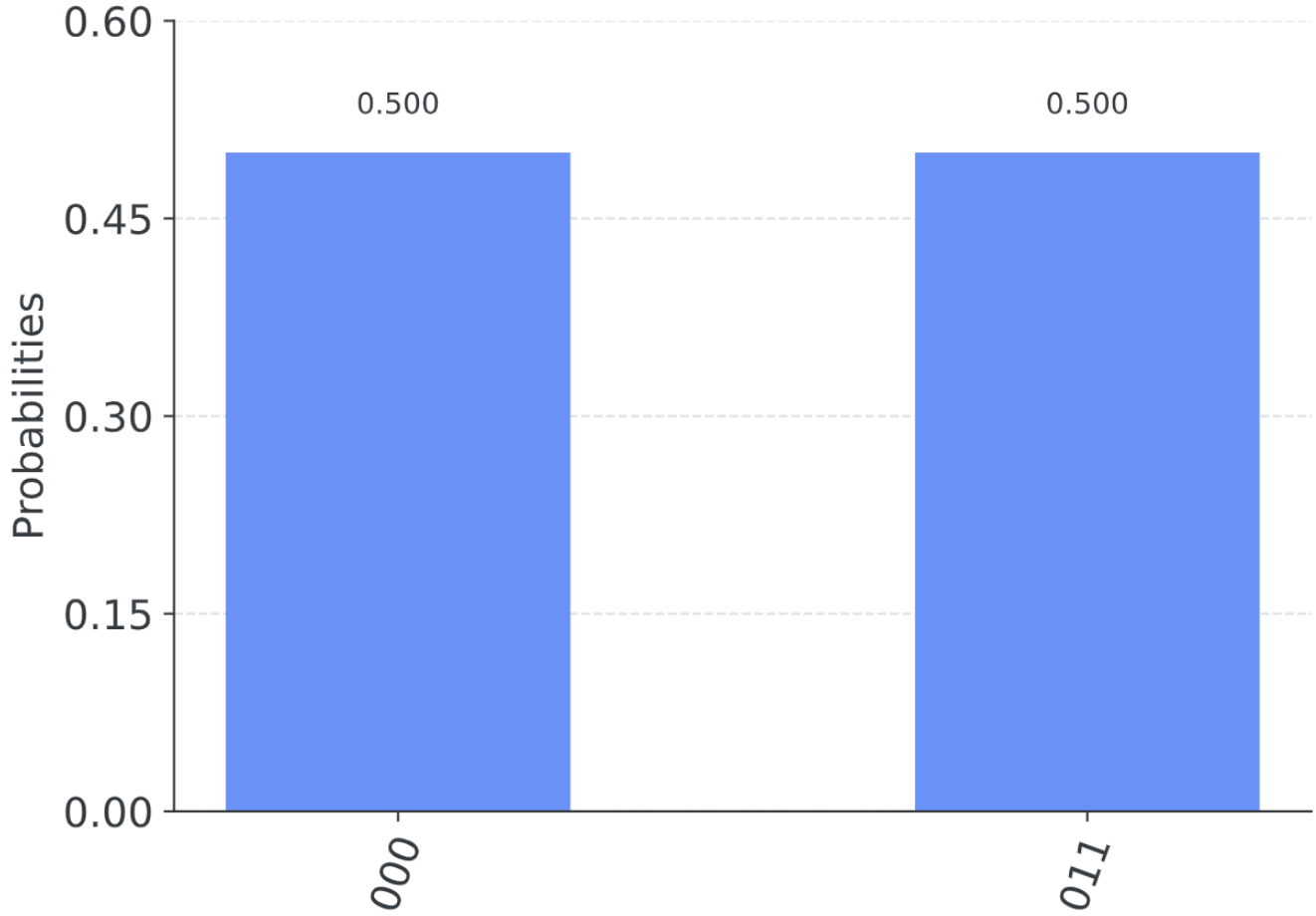





```
Statevector([0.        +0.j, 0.        +0.j, 0.70710678+0.j,
             0.70710678+0.j],
            dims=(2, 2))
```

```
#CASO 5.
import math
desired_vector=[2/math.sqrt(8)
,0
,0
,2/math.sqrt(8)
,0
,0
,0
,0
]
q=QuantumRegister(3)
qc=QuantumCircuit(q)
qc.initialize(desired_vector, [q[0],q[1],q[2]])
qc.draw()
```






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

ACTIVIDADES A MANO

1) If we measure q1 and the result is 1. What will be the result of measuring q0(after measuring q1)?

Será un 1 porque está entrelazado.

---

2) If we now repeat the same measurement experiment for D. What will be the result of measuring q0 if we have previouly measure q1 and have got a 1? Será 1 también.

---

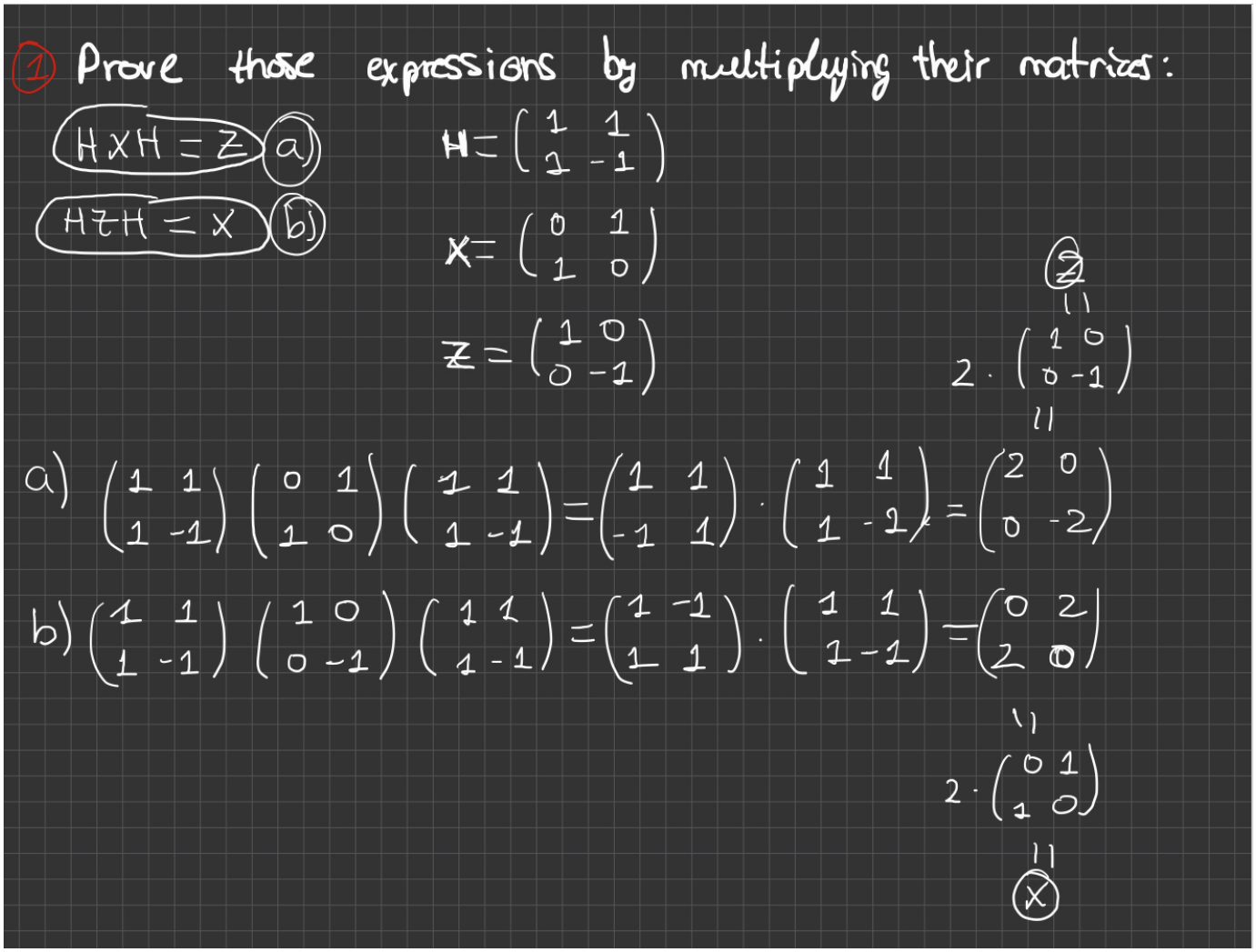
3) Is any of those states and an entangled state? Prove that by trying to write them as two separate qubit states.

C está entrelazado porque en el momento en el que medimos un 0 en q0 o q1 sabemos sí o sí que el otro estado va a ser el mismo que hayamos medido. Sin embargo, esto no sucede con D.

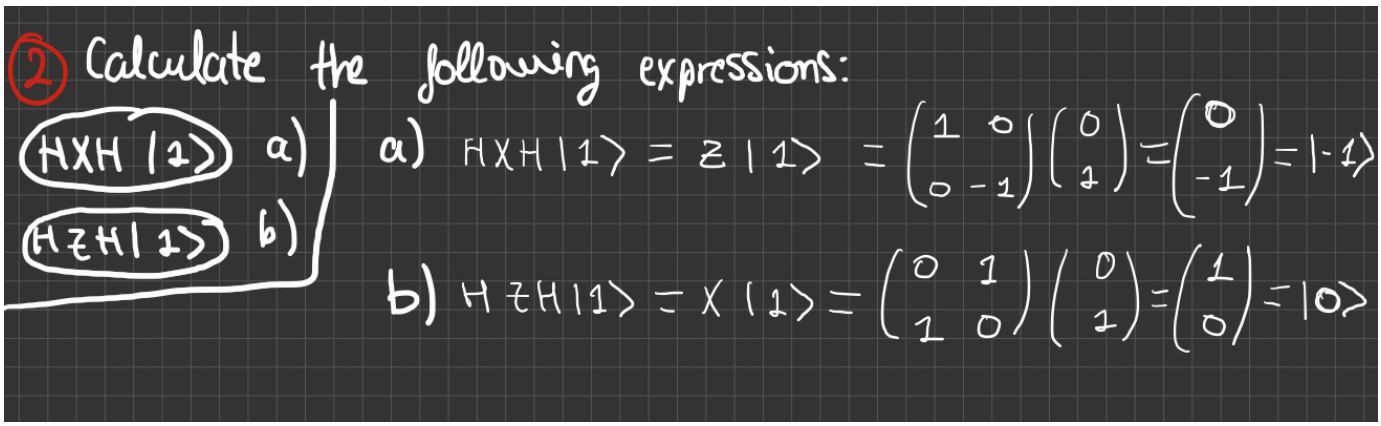
PARTE 2

ACTIVIDADES A MANO

1.- PROVE THESE EXPRESSIONS BY MULTIPLYING THEIR CORRESPONDING MATRICES.



2.- CALCULATE THE FOLLOWING EXPRESSIONS:



1.- CREAR LOS CIRCUITOS HXH Y HZH

CIRCUITO HXH

```
qc = QuantumCircuit(1)
#### your code goes here

qc.x(0)
qc.h(0)
qc.x(0)
qc.h(0)

qc.draw()
```

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

```
Statevector([-6.123234e-17+7.49879891e-33j, -1.000000e+00+1.22464680e-16j],
            dims=(2,))
```


```
qc = QuantumCircuit(1)
#### your code goes here
qc.x(0)
qc.z(0)
qc.draw()
```

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

```
Statevector([ 6.123234e-17+1.49975978e-32j, -1.000000e+00-3.67394040e-16j],
            dims=(2,))
```

CIRCUITO HZH

```
qc = QuantumCircuit(1)
#### your code goes here

qc.x(0)
qc.h(0)
qc.z(0)
qc.h(0)

qc.draw()
```

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

```
Statevector([1.+0.j, 0.+0.j],
            dims=(2,))
```

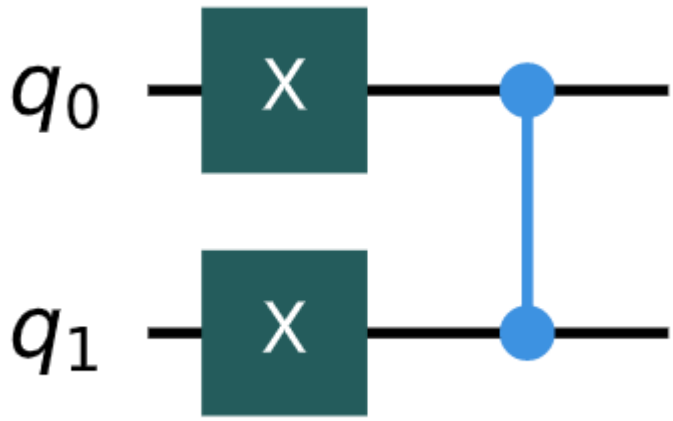

```
qc = QuantumCircuit(1)
#### your code goes here
qc.x(0)
qc.x(0)
qc.draw()
```

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


```
Statevector([1.+0.j, 0.+0.j],
            dims=(2,))
```

1.2.- HACIENDO CZ CON CNOT

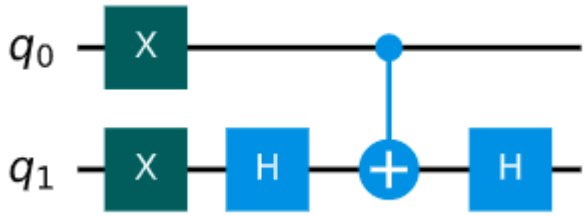
```
qc = QuantumCircuit(2)

#### your code goes here
qc.x(0)
qc.x(1)
qc.cz(0,1)

qc.draw()
```




```
Statevector([ 0.+0.j,  0.+0.j,  0.+0.j, -1.+0.j],
            dims=(2, 2))
```

```
qc = QuantumCircuit(2)

#### your code goes here
qc.x(0)
qc.x(1)
qc.h(1)
qc.cnot(0,1)
qc.h(1)



qc.draw()


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

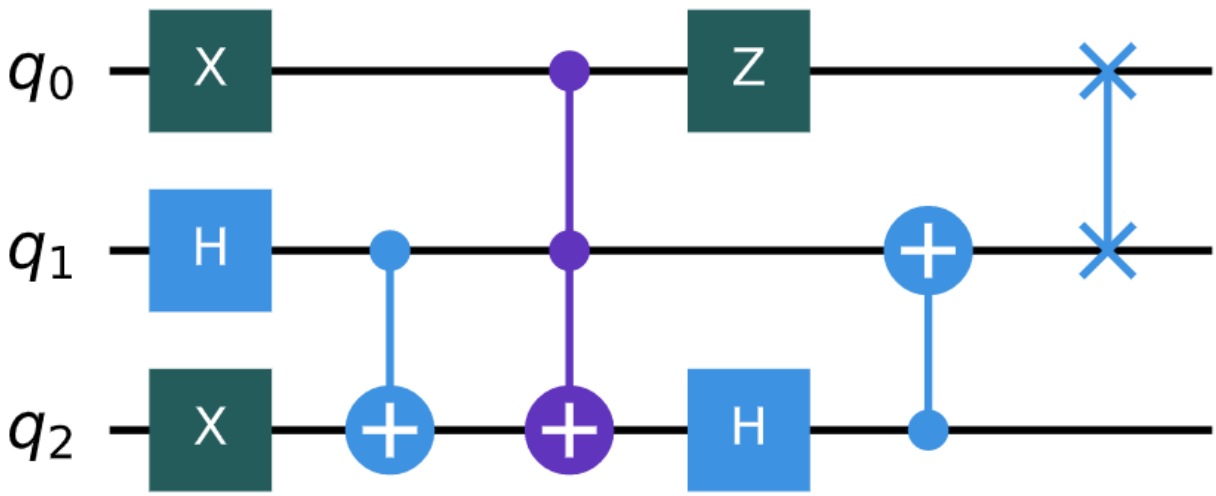
```
Statevector([ 0.+0.00000000e+00j,  0.-1.22464680e-16j,  0.+0.00000000e+00j,
             -1.+2.46519033e-32j],
            dims=(2, 2))
```



PARTE 3

1.- CALCULAR EL ESTADO DE SALIDA DE UN CIRCUITO CUÁNTICO

```
qc = QuantumCircuit(3)

#### your code goes here

qc.x(0)
qc.h(1)
qc.x(2)
qc.cx(1,2)
qc.ccx(0,1,2)
qc.z(0)
qc.h(2)
qc.cnot(2,1)
qc.swap(0,1)


qc.draw()
```




```
backend = Aer.get_backend('statevector_simulator') # Tell Qiskit how to simulate ou
# Let's display the output state vector
```

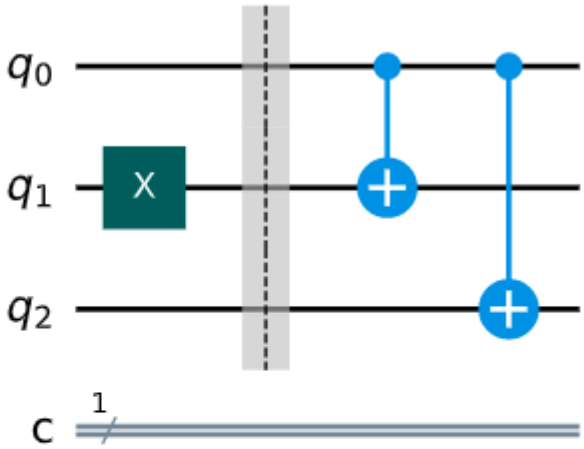
```
result = execute(qc,backend).result() # Do the simulation, returning the result
out_state = result.get_statevector()
print(out_state) # Display the output state vector
```

```
Statevector([ 0. +0.000000e+00j,  0. +0.000000e+00j, -0.5+6.123234e-17j,
             -0.5+6.123234e-17j,  0. +0.000000e+00j,  0. +0.000000e+00j,
              0.5-6.123234e-17j,  0.5-6.123234e-17j],
            dims=(2, 2, 2))
```

2.- CREANDO UN SUMADOR QUÁNTICO

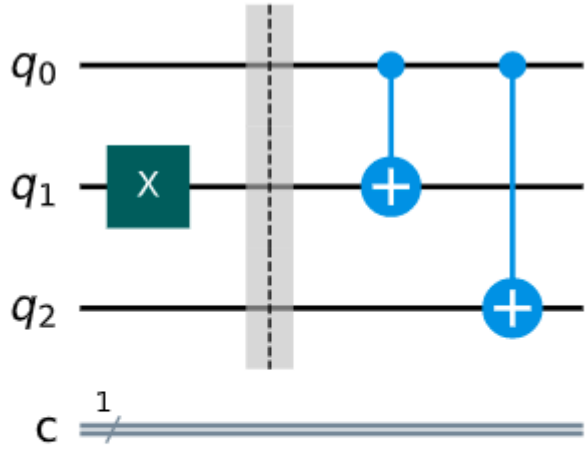
2.1- EL CIRCUITO SUM

```
# this is where your program for the SUM circuit goes
qc = QuantumCircuit(3,1)



#initialization of qubits


# barrier between input state and gate operation
qc.barrier()
qc.cx(0,1)
qc.cx(0,2)

#gates that perfrom the addition



display(qc.draw())

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




```
Statevector([0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j],
            dims=(2, 2, 2))
```

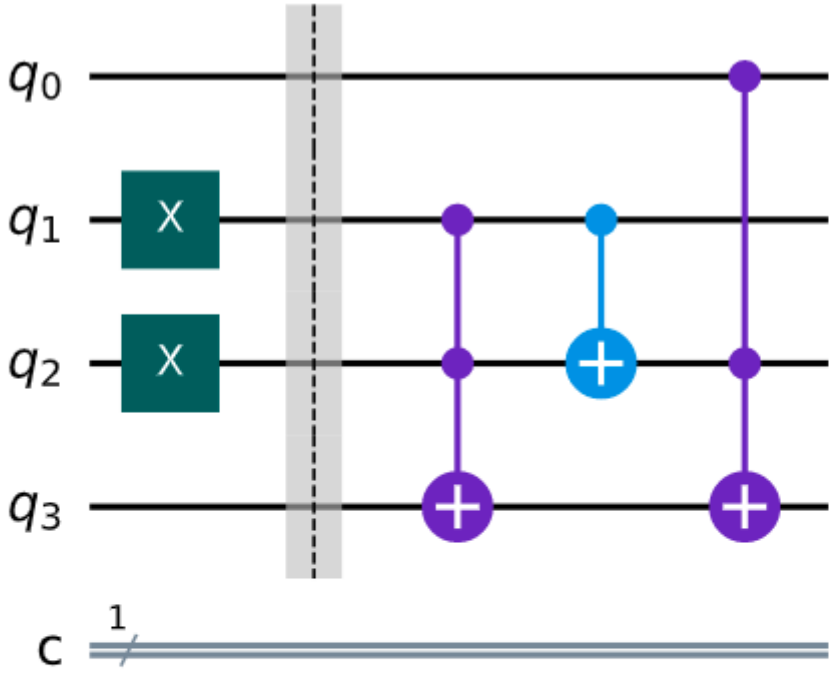
```
# this is where your program for the SUM circuit goes
qc = QuantumCircuit(3,1)



#initialization of qubits
qc.x(1)

# barrier between input state and gate operation
qc.barrier()
qc.cx(0,1)
qc.cx(0,2)

#gates that perfrom the addition



display(qc.draw())

backend = Aer.get_backend('statevector_simulator') # Tell Qiskit how to simulate ou
# Let's display the output state vector
result = execute(qc,backend).result() # Do the simulation, returning the result
out_state = result.get_statevector()
print("SALIDA DEL OUTPUT: ", output)
```




```
SALIDA DEL OUTPUT:  0
```

2.2 EL CIRCUITO CARRY

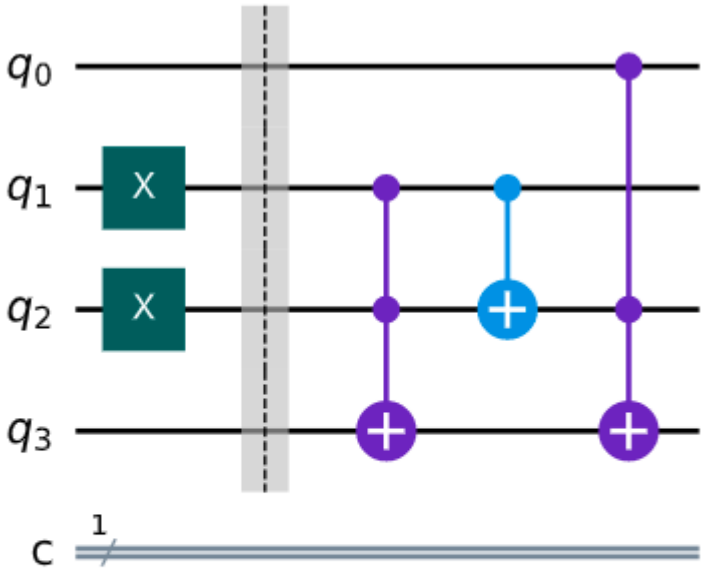
```
# this is where your program for the CARRY circuit goes

qc = QuantumCircuit(4,1)


#initialization of qubits
qc.x(1)
qc.x(2)

# barrier between input state and gate operation
qc.barrier()


# gates that perform the Carry
qc.ccx(1,2,3)
qc.cx(1,2)
qc.ccx(0,2,3)

display(qc.draw())


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




```
Statevector([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j],
            dims=(2, 2, 2, 2))
```

```
# Measuring q2 and displaying the result
# this is where your program for the CARRY circuit goes

qc = QuantumCircuit(4,1)


#initialization of qubits
qc.x(1)
qc.x(2)

# barrier between input state and gate operation
qc.barrier()


# gates that perform the Carry
qc.ccx(1,2,3)
qc.cx(1,2)
qc.ccx(0,2,3)

display(qc.draw())
```

```
qc.measure(3,0)
qc.draw()

# We'll run the program on a simulator
backend = Aer.get_backend('qasm_simulator')
# Since the output will be deterministic, we can use just a single shot to get it
job = execute(qc, backend, shots=1, memory=True)
output = job.result().get_memory()[0]
print("SALIDA DEL OUTPUT: ", output)
```



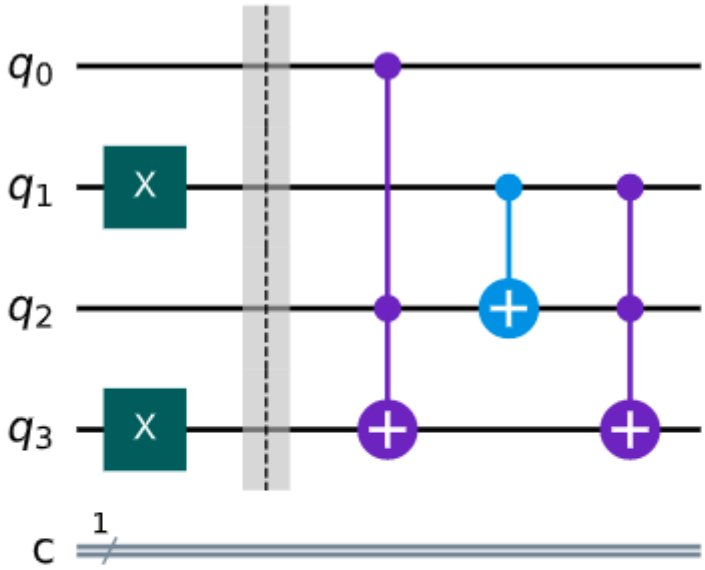

```
SALIDA DEL OUTPUT:  1
```

2.2.2.- EL CIRCUITO RCARRY

```
# this is where your program for the RCARRY circuit goes

qc = QuantumCircuit(4,1)


#initialization of qubits
qc.x(1)
qc.x(3)

# barrier between input state and gate operation
qc.barrier()


# gates that perform the Carry
qc.ccx(0,2,3)
qc.cx(1,2)
qc.ccx(1,2,3)
```

```
display(qc.draw())
```

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



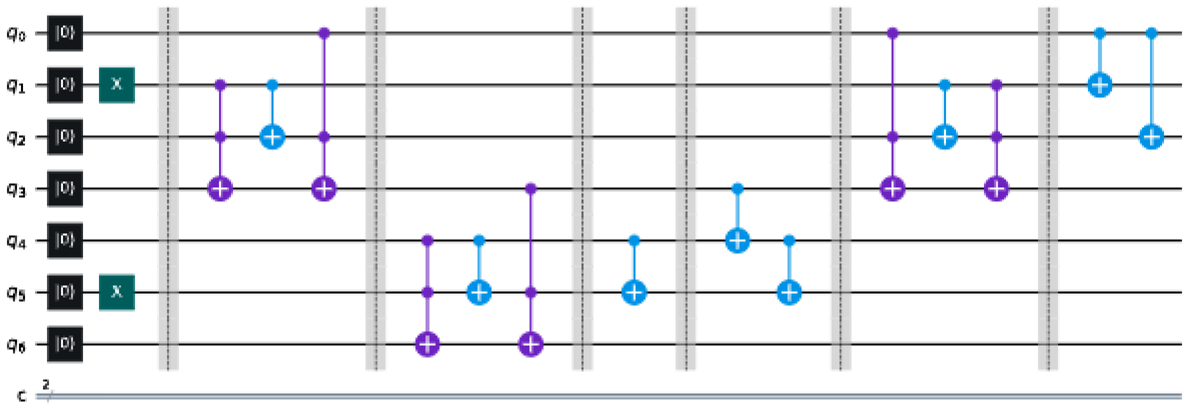

```
Statevector([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j],
            dims=(2, 2, 2, 2))
```

2.3.- CREACIÓN DE UN SUMADOR DE 2 BITS

```
## Your code goes here

# This is where your program for the plain 2-bits adder circuit goes

qc = QuantumCircuit(7,2)
qc.reset(range(7))

## Initialise the operands
qc.x(1)
qc.x(5)

qc.barrier()
```

```
## carry, CNOT, Sum and rcarry blocks
qc.ccx(1,2,3)
qc.cx(1,2)
qc.ccx(0,2,3)
qc.barrier()
qc.ccx(4,5,6)
qc.cx(4,5)
qc.ccx(3,5,6)
qc.barrier()
qc.cx(4,5)
qc.barrier()
qc.cx(3,4)
qc.cx(4,5)
qc.barrier()
qc.ccx(0,2,3)
qc.cx(1,2)
qc.ccx(1,2,3)
qc.barrier()
qc.cx(0,1)
qc.cx(0,2)

display(qc.draw())
```

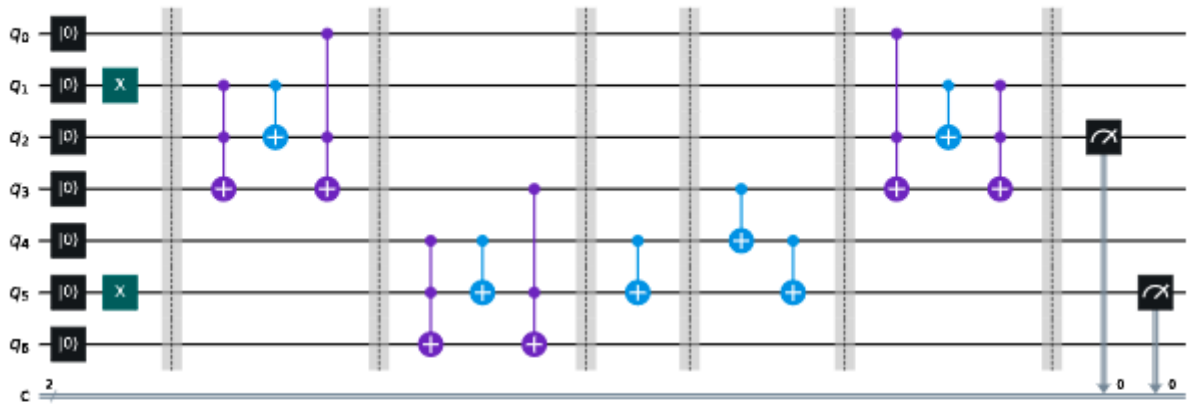
```
backend = Aer.get_backend('statevector_simulator') # Tell Qiskit how to simulate ou
# Let's display the output state vector
result = execute(qc,backend).result() # Do the simulation, returning the result
out_state = result.get_statevector()
print(out_state) # Display the output state vector
```



```
Statevector([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j],
            dims=(2, 2, 2, 2, 2, 2, 2))
```

```
# Measuring qubits and displaying the result of the addition
## Your code goes here

# This is where your program for the plain 2-bits adder circuit goes

qc = QuantumCircuit(7,2)
qc.reset(range(7))

## Initialise the operands
qc.x(1)
qc.x(5)

qc.barrier()

## carry, CNOT, Sum and rcarry blocks
qc.ccx(1,2,3)
```

```
qc.cx(1,2)
qc.ccx(0,2,3)
qc.barrier()
qc.ccx(4,5,6)
qc.cx(4,5)
qc.ccx(3,5,6)
qc.barrier()
qc.cx(4,5)
qc.barrier()
qc.cx(3,4)
qc.cx(4,5)
qc.barrier()
qc.ccx(0,2,3)
qc.cx(1,2)
qc.ccx(1,2,3)
qc.barrier()
qc.cx

qc.measure(2,0)
qc.measure(5,0)
display(qc.draw())

# We'll run the program on a simulator
backend = Aer.get_backend('qasm_simulator')
# Since the output will be deterministic, we can use just a single shot to get it
job = execute(qc, backend, shots=1, memory=True)
output = job.result().get_memory()[0]
print("SALIDA DEL OUTPUT: ", output)
```




```
SALIDA DEL OUTPUT:  01
```