

# QISKit

Quantum Information Science Kit  
Open Source Code

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## ESFERA de BLOCH

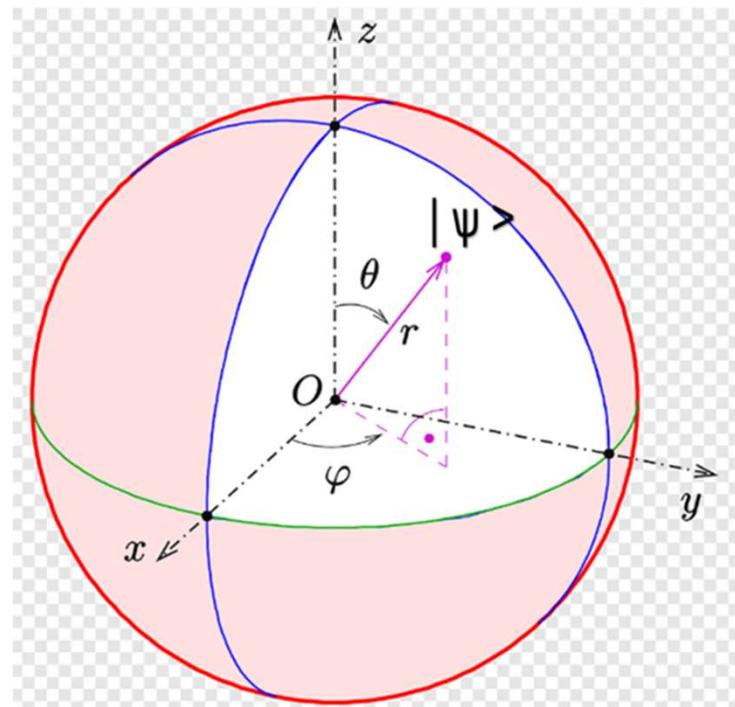
$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\psi\rangle = \cos(\Theta/2)|0\rangle + e^{i\varphi}\sin(\Theta/2)|1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1$$

$$\begin{cases} \alpha = \cos(\Theta/2) \\ \beta = e^{i\varphi}\sin(\Theta/2) \end{cases}$$

Espacio Hilbert  
 $n = 2$   
Base  
 $\{|0\rangle, |1\rangle\}$   
Qbits  $\in$  esp. Hilbert



$$\begin{aligned} |\alpha|^2 &= \cos^2(\Theta/2) \\ |\beta|^2 &= \sin^2(\Theta/2) \\ |\alpha|^2 + |\beta|^2 &= \cos^2(\Theta/2) + \sin^2(\Theta/2) \end{aligned}$$

## Creación de un circuito simple y visualización de la esfera de Bloch

```
#####
# CURSO QISKIT 2025
# Crear circuito simple y visualizar esfera de Bloch

from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1,1)
stateVectorResult = Statevector(qc)
plot_bloch_multivector(stateVectorResult)

#####
```

## ESFERA de BLOCH

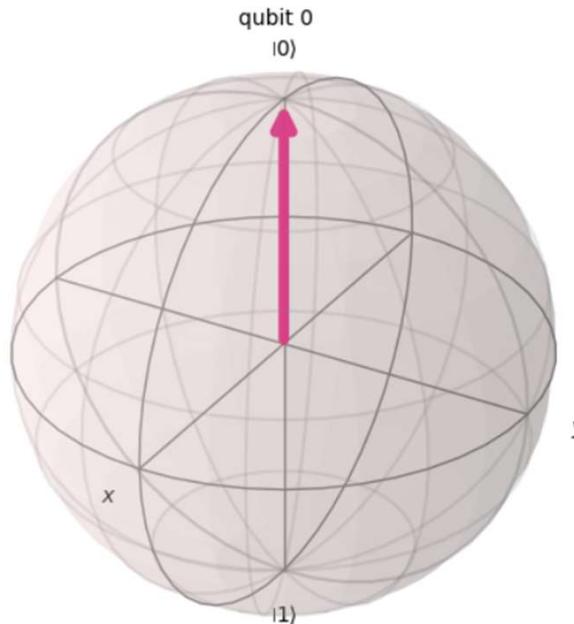
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Creación de un circuito simple y visualización de la esfera de Bloch

```
[1]: from qiskit import QuantumCircuit  
from qiskit.primitives import StatevectorSampler  
from qiskit.quantum_info import Statevector  
from qiskit.visualization import *  
  
qc = QuantumCircuit(1,1)  
stateVectorResult = Statevector(qc)  
plot_bloch_multivector(stateVectorResult)
```

[1]:



HADAMARD

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Hadamard Gate

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
$$|\psi\rangle \xrightarrow{\text{H}} |\psi_s\rangle$$

Hadamard

$$|\psi_s\rangle = H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = |+\rangle$$

↑  
|0>

HADAMARD

## Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Incorporación compuerta Hadamard y poner en superposición al qubit

```
#####
# CURSO QISKIT 2025
# Crear circuito simple con compuerta HADAMARD

from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1,1)

# agregamos compuerta Hadamard H

qc.h(0)
qc.draw(output='mpl')

#####
```

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

HADAMARD

Incorporación compuerta Hadamard y poner en superposición al qubit

[3]:

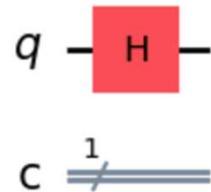
```
from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSimulator
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1,1)

# agregamos compuerta Hadamard H

qc.h(0)
qc.draw(output='mpl')
```

[3]:



HADAMARD

## Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Incorporación compuerta Hadamard y poner en superposición al qubit

```
#####
# CURSO QISKIT 2025
# Crear circuito simple y visualizar esfera de Bloch

from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1,1)

# agregamos compuerta Hadamard H

qc.h(0)
stateVectorResult = Statevector(qc)
plot_bloch_multivector(stateVectorResult)
#####
```

HADAMARD

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Incorporación compuerta Hadamard y poner en superposición al qubit

```
[4]: #####
# CURSO QISKIT 2025
# Crear circuito simple y visualizar esfera de Bloch

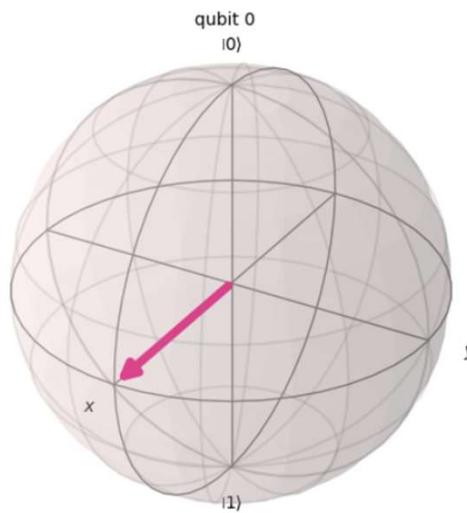
from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1,1)

# agregamos compuerta Hadamard H

qc.h(0)
stateVectorResult = Statevector(qc)
plot_bloch_multivector(stateVectorResult)
#####
```

[4]:



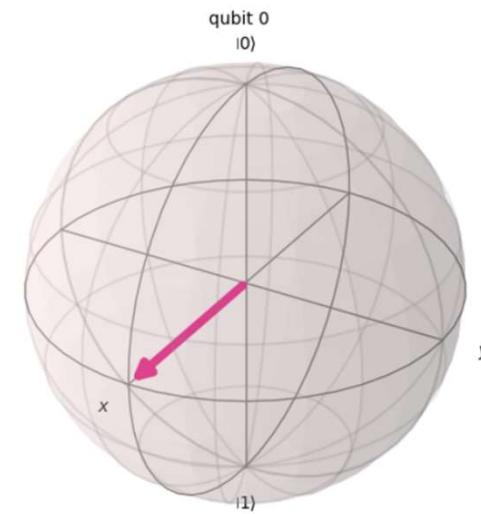
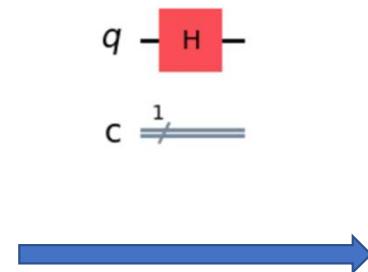
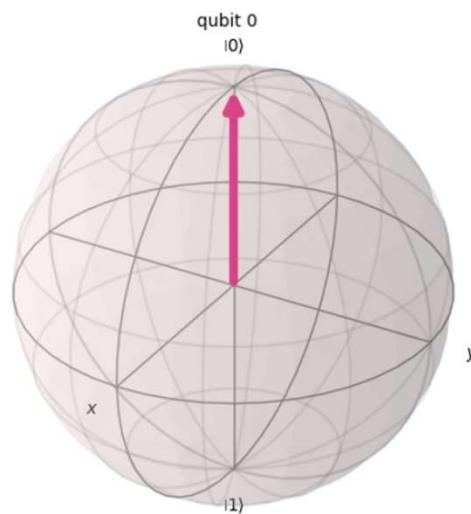
$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$

HADAMARD

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Incorporación compuerta Hadamard y poner en superposición al qubit



$|0\rangle$

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$

GATE X

## Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

### Incorporación compuerta X

```
#####
# CURSO QISKIT 2025
# Hadamard + X

qc = QuantumCircuit(1)

# agregamos compuerta X

qc.x(0)
qc.h(0)
qc.draw(output='mpl')
#####
```

GATE X

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Incorporación compuerta X

```
[6]: # CURSO QISKIT 2025
# Hadamard + X

qc = QuantumCircuit(1)

# agregamos compuerta X

qc.x(0)

qc.h(0)

qc.draw(output='mpl')
```

[6]:



GATE X

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Incorporación compuerta X Esfera Bloch

```
#####
# CURSO QISKIT 2025
# Hadamard + X

qc = QuantumCircuit(1)

# agregamos compuerta X

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

stateVectorResult = Statevector(qc)
plot_bloch_multivector(stateVectorResult)
#####|
```

GATE X

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Incorporación compuerta X Esfera Bloch

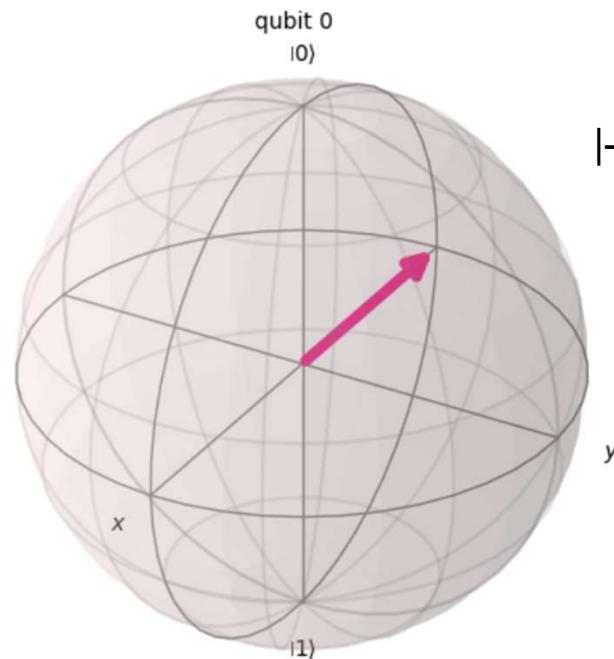
```
[7]: qc = QuantumCircuit(1)

# agregamos compuerta X

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

stateVectorResult = statevector(qc)
plot_bloch_multivector(stateVectorResult)
```

[7]:



$$|-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

14



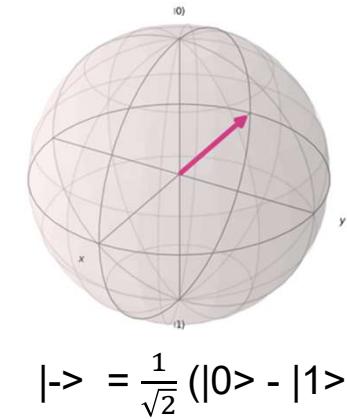
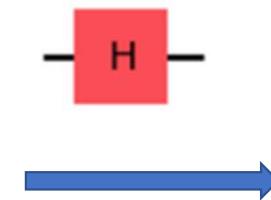
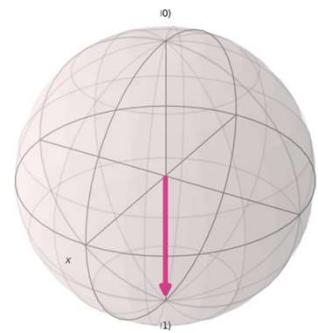
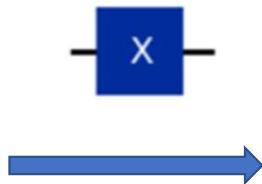
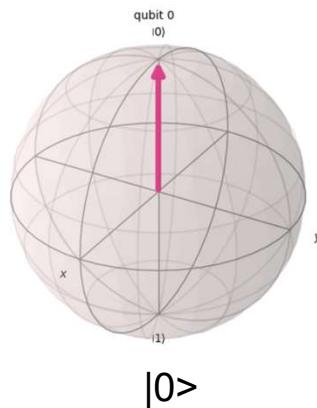
CamScanner

GATE X

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Incorporación compuerta X+H Esfera Bloch



$$|-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

MEDICION

## Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

### Incorporación Medidor (NO Reversible)

```
#####
# CURSO QISKIT 2025
# Medidor

qc = QuantumCircuit(1,1)

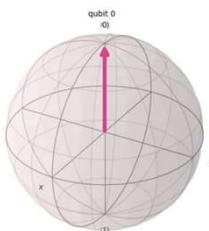
# agregamos Medicion

qc.h(0)
qc.measure(0,0)
qc.draw(output='mpl')
#####
```

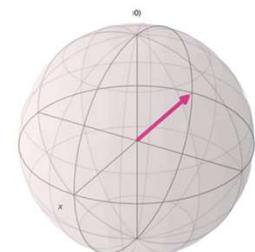
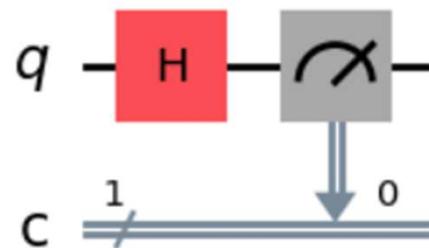
## Incorporación Medidor

```
[17]: qc = QuantumCircuit(1,1)  
  
# agregamos Medicion  
  
qc.h(0)  
qc.measure(0,0)  
qc.draw(output='mpl')
```

```
[17]:
```



$|0\rangle$



$|->$

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

HISTOGRAMA

## HISTOGRAMA

```
#####
# CURSO QISKIT 2025
# HISTORGRAMA

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(1,1)

qc.h(0)
qc.measure(0,0)

simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
sim_result = simulator.run(compiled_circuit, shots= 100, memory=True).result()
counts = sim_result.get_counts()
print(counts)
plot_histogram(counts)

#####
```

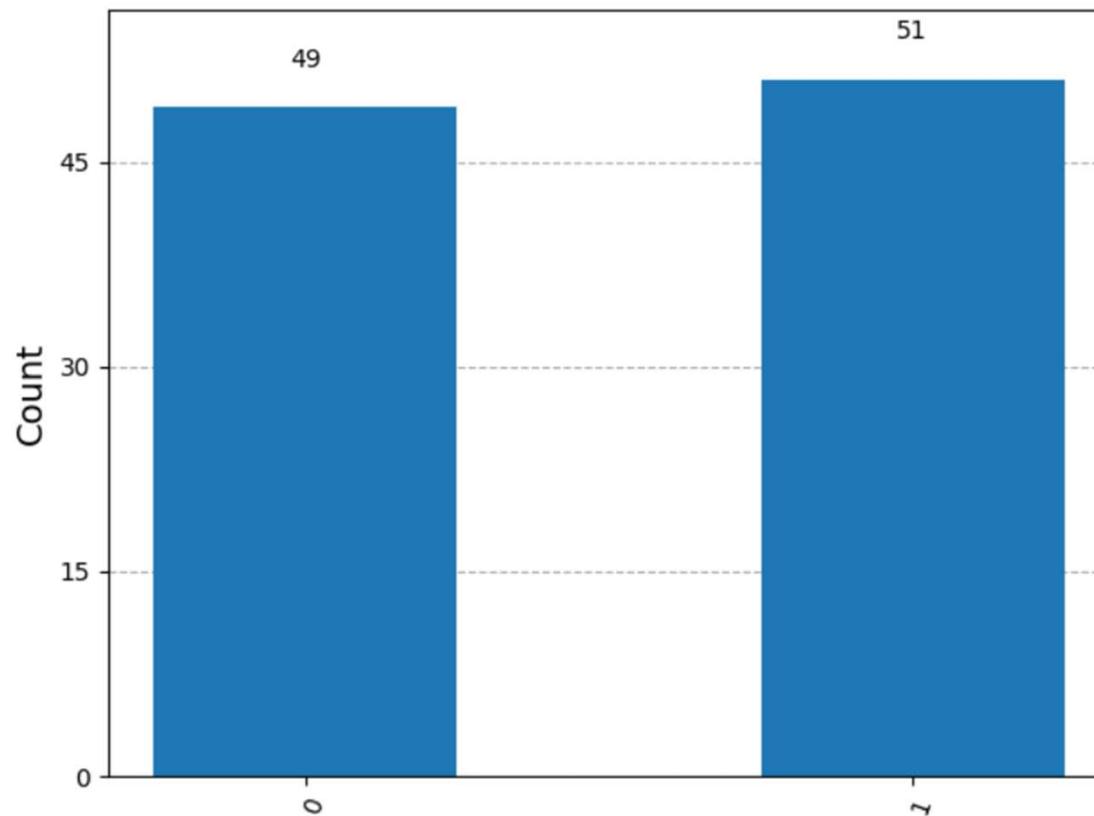
# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

HISTOGRAMA

HISTOGRAMA

{'0': 49, '1': 51}



Simulación #1

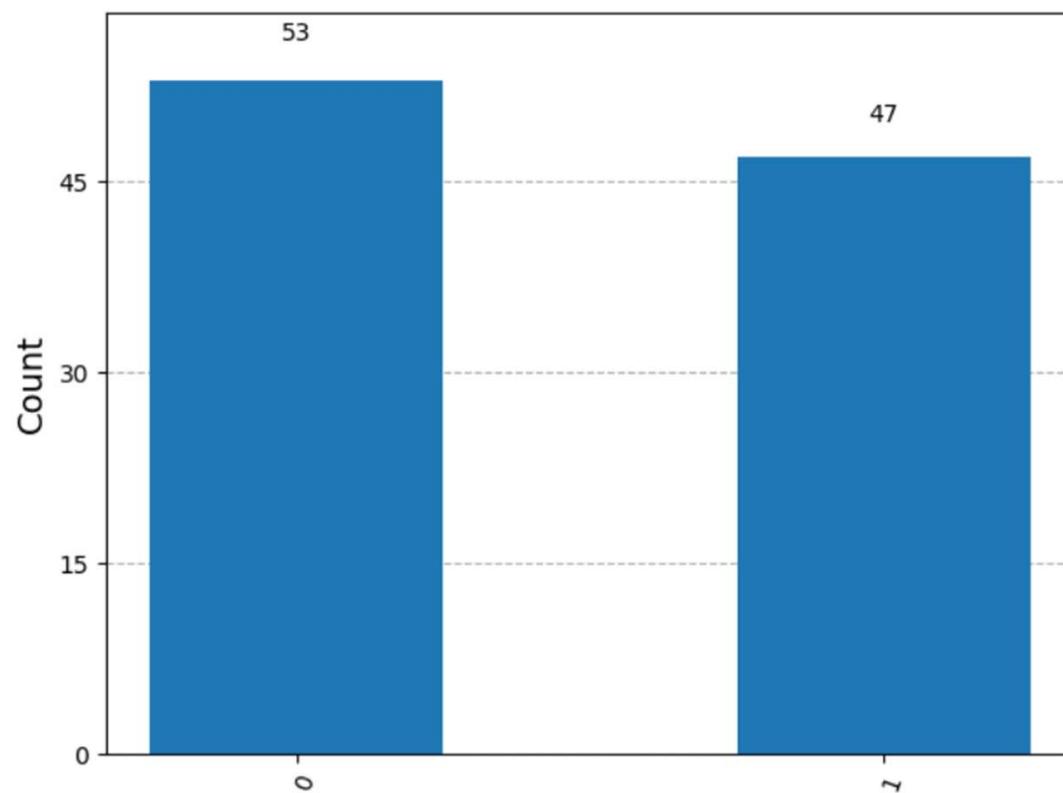
# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

HISTOGRAMA

HISTOGRAMA

{'0': 53, '1': 47}



## HISTOGRAMA

### HISTOGRAMA

```
#####
# CURSO QISKIT 2025
# HISTOGRAMA

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.h(0)
qc.h(1)
qc.measure([0,1],[0,1])
qc.draw(output='mpl')

#####
```

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## HISTOGRAMA

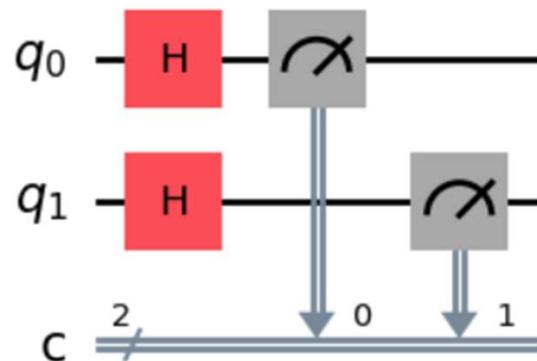
### HISTOGRAMA

```
[1]: from qiskit import QuantumCircuit
      from qiskit import transpile
      from qiskit_aer import AerSimulator
      from qiskit.visualization import plot_histogram

      qc = QuantumCircuit(2,2)

      qc.h(0)
      qc.h(1)
      qc.measure([0,1],[0,1])
      qc.draw(output='mpl')
```

[1]:



## HISTOGRAMA

### HISTOGRAMA

```
#####
# CURSO QISKIT 2025
# HISTOGRAMA

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.h(0)
qc.h(1)
qc.measure([0,1],[0,1])
qc.draw(output='mpl')
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
sim_result = simulator.run(compiled_circuit, shots= 1000, memory=True).result()
counts = sim_result.get_counts()
print(counts)
plot_histogram(counts)

#####
```

# Quantum Computing QISKit

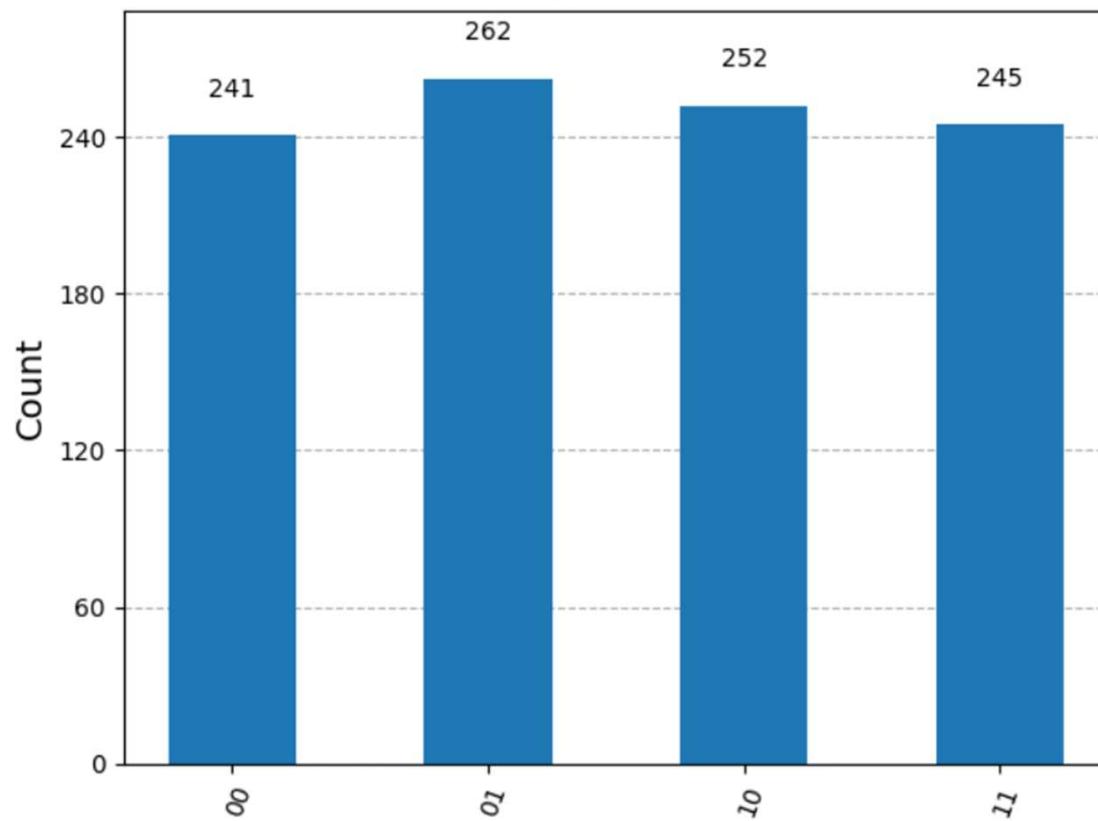
Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

HISTOGRAMA

HISTOGRAMA

```
{'00': 241, '01': 262, '11': 245, '10': 252}
```

[3]:



Simulación #1  
Shots = 1.000

# Quantum Computing QISKit

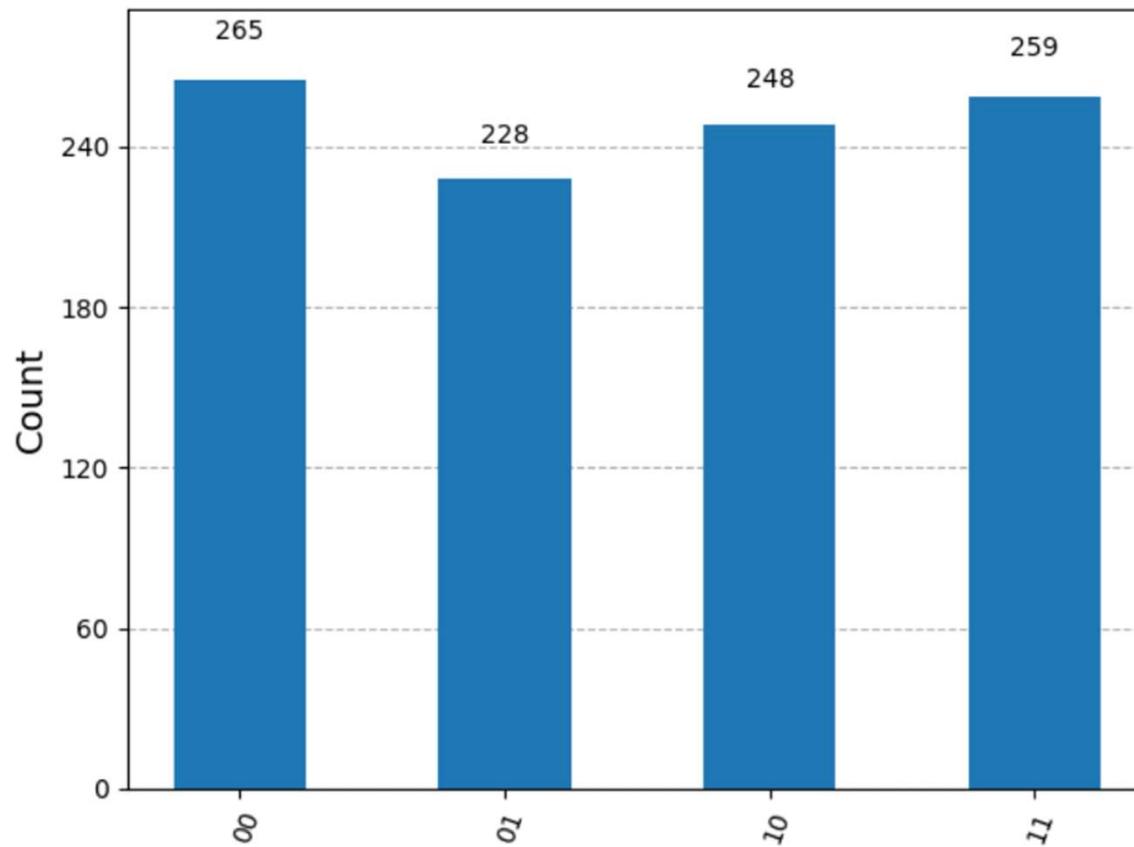
Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

HISTOGRAMA

HISTOGRAMA

```
{'00': 265, '11': 259, '10': 248, '01': 228}
```

```
[4]:
```



Simulación #2  
Shots = 1.000

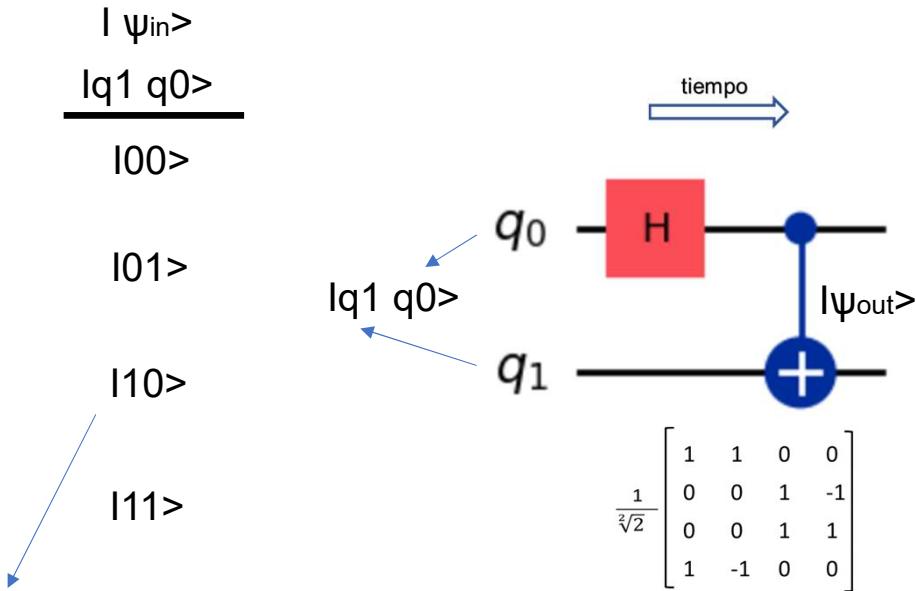
## ENTAGLEMENT

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Estados de Bell

$$|\psi_{in}\rangle \neq |q_1\rangle \otimes |q_2\rangle$$



$$|\psi_{out}\rangle$$

Qbits - Entrelazados

$$\frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) = |\Phi^+\rangle$$

$$\frac{1}{\sqrt{2}} (|00\rangle - |11\rangle) = |\Phi^-\rangle$$

$$\frac{1}{\sqrt{2}} (|01\rangle + |10\rangle) = |\Psi^+\rangle$$

$$\frac{1}{\sqrt{2}} (|01\rangle - |10\rangle) = |\Psi^-\rangle$$

Estados Bell

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix} |10\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = |\Psi^+\rangle$$

Estado de Bell  $|\phi^+\rangle$ 

```
#####
# CURSO QISKIT 2025
# ENTAGLEMENT

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.h(0)
qc.cx(0,1)
qc.measure([0,1],[0,1])
qc.draw(output='mpl')

#####
```

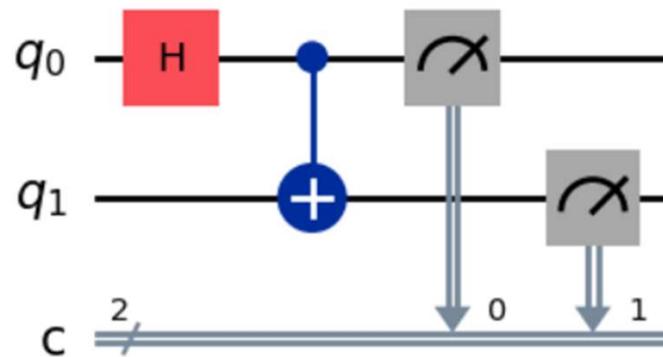
Estado de Bell  $|\emptyset^+\rangle$ 

```
[2]: from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.h(0)
qc.cx(0,1)
qc.measure([0,1],[0,1])
qc.draw(output='mpl')
```

```
[2]:
```



$$|\emptyset^+\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$$

Estado de Bell  $|\phi^+\rangle$ 

```
#####
# CURSO QISKIT 2025
# ENTAGLEMENT

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.h(0)
qc.cx(0,1)
qc.measure([0,1],[0,1])

simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
sim_result = simulator.run(compiled_circuit, shots= 1000, memory=True).result()
counts = sim_result.get_counts()
print(counts)
plot_histogram(counts)
#####
```

## ENTAGLEMENT

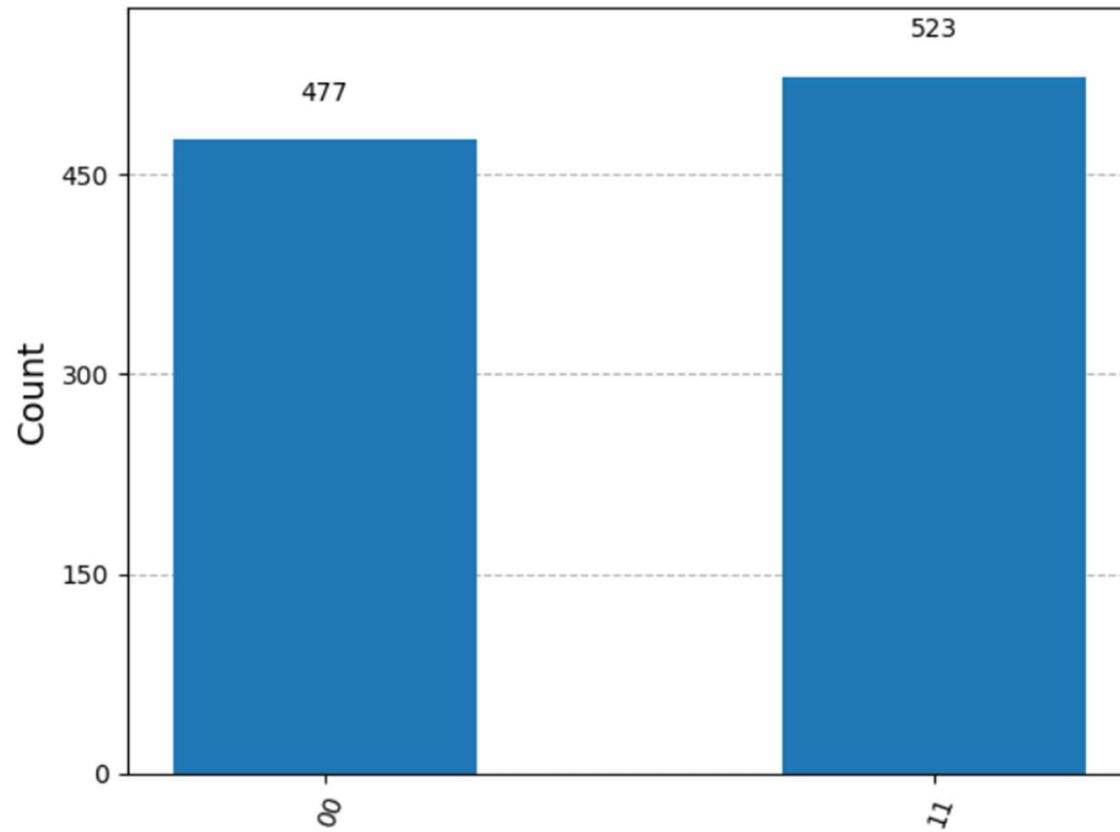
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Estado de Bell  $|\phi^+\rangle$

{'00': 477, '11': 523}

[4]:



## ENTAGLEMENT

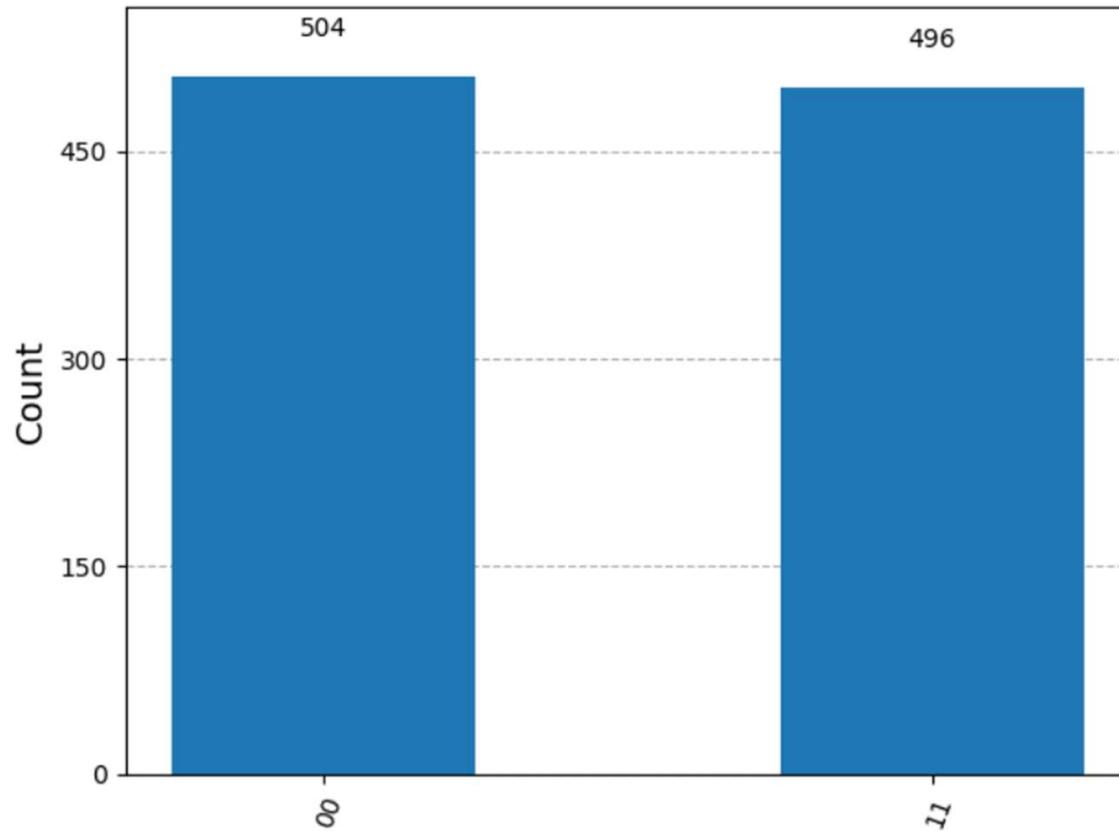
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Estado de Bell  $|\phi^+\rangle$

{'11': 496, '00': 504}

[7]:



Estado de Bell  $|\Psi^+\rangle$ 

```
#####
# CURSO QISKIT 2025
# ENTAGLEMENT

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.x(1)
qc.barrier()
qc.h(0)
qc.cx(0,1)
qc.barrier()
qc.measure([0,1],[0,1])

qc.draw(output='mpl')
#####
```

## ENTAGLEMENT

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

### Estado de Bell $|\Psi^+\rangle$

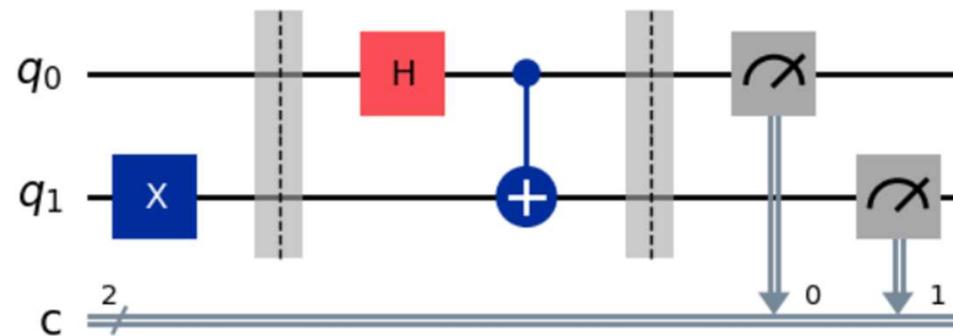
```
[2]: from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.x(1)
qc.barrier()
qc.h(0)
qc.cx(0,1)
qc.barrier()
qc.measure([0,1],[0,1])

qc.draw(output='mpl')
```

[2]:



$$|\Psi^+\rangle = \frac{1}{\sqrt{2}} (|01\rangle + |10\rangle)$$

Estado de Bell  $|\Psi^+\rangle$ 

```
#####
# CURSO QISKIT 2025
# ENTAGLEMENT

from qiskit import QuantumCircuit
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2,2)

qc.x(1)
qc.barrier()
qc.h(0)
qc.cx(0,1)
qc.barrier()
qc.measure([0,1],[0,1])

simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
sim_result = simulator.run(compiled_circuit, shots= 1000, memory=True).result()
counts = sim_result.get_counts()
print(counts)
plot_histogram(counts)

#####
```

## Representación QUBIT con BLOCH

```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

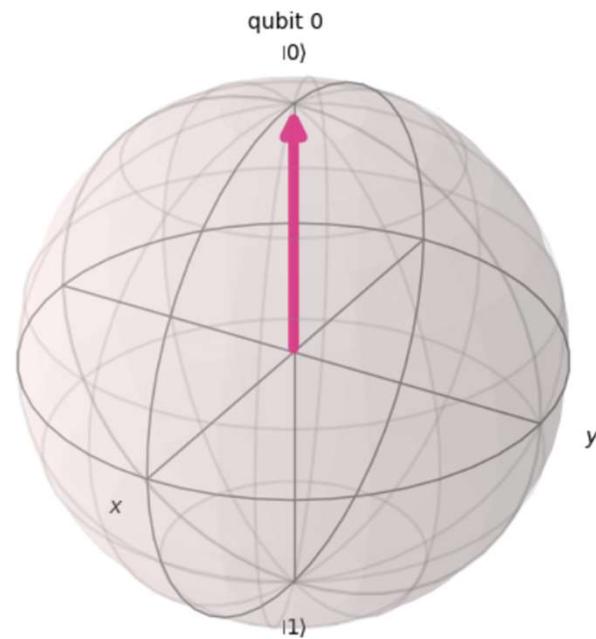
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('bloch')
#####
```

## Representación QUBIT con BLOCH

```
[4]: from qiskit import QuantumCircuit  
from qiskit.quantum_info import Statevector  
from qiskit.visualization import *  
  
qc = QuantumCircuit(1)  
stateVectorResult = Statevector(qc)  
print('state vector results', stateVectorResult)  
stateVectorResult.draw('bloch')  
  
state vector results Statevector([1.+0.j, 0.+0.j]),  
dims=(2,))
```

[4]:



## Representación QUBIT con QSHERE

```
#####
# CURSO QISKIT 2025
# Esfera de QSHERE

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('qsphere')
#####
```

**QUBIT**

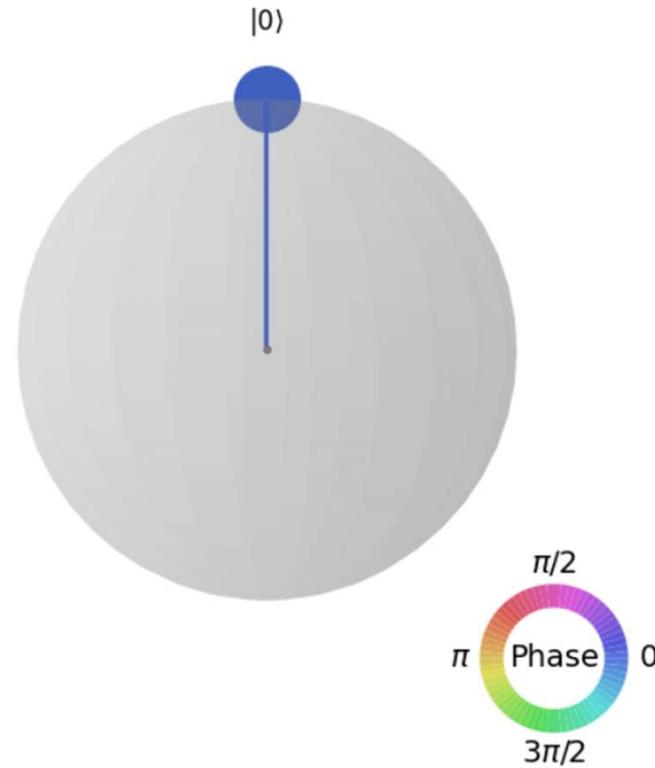
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Representación QUBIT con QSHERE

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

[6]:



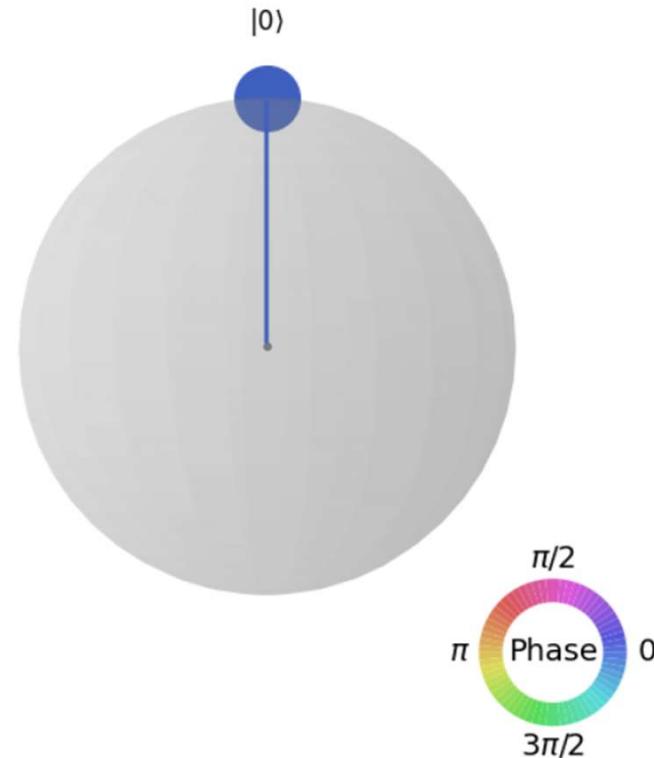
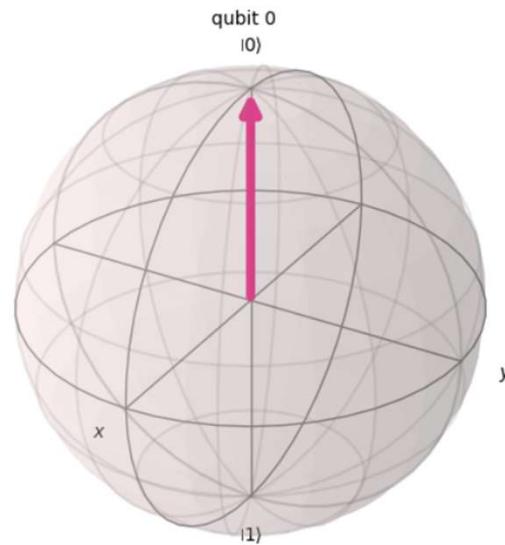
**QUBIT**

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Representación QUBIT  $|0\rangle$

[4]:



## Representación QUBIT |1&gt;

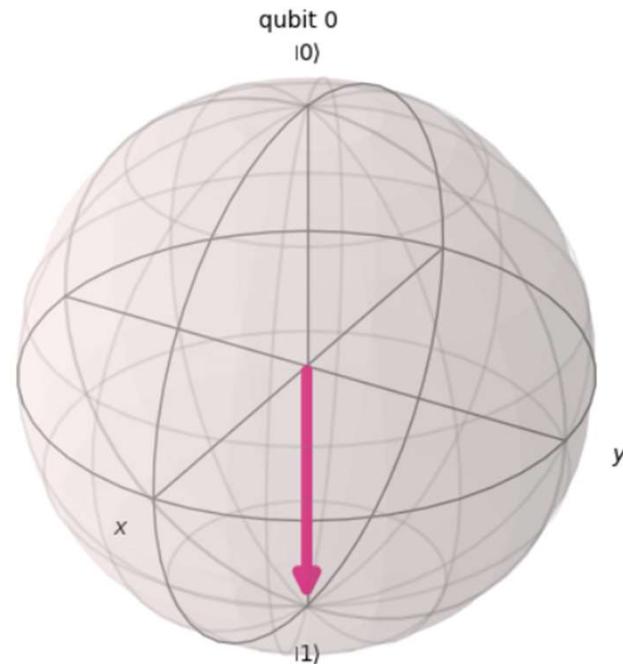
```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
qc.x(0)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('bloch')
#####
```

Representación QUBIT  $|1\rangle$ 

```
state vector results Statevector([0.+0.j, 1.+0.j],  
                               dims=(2,))  
[8]:
```



## Representación QUBIT |1&gt;

```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
qc.x(0)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('qsphere')
#####
```

QUBIT

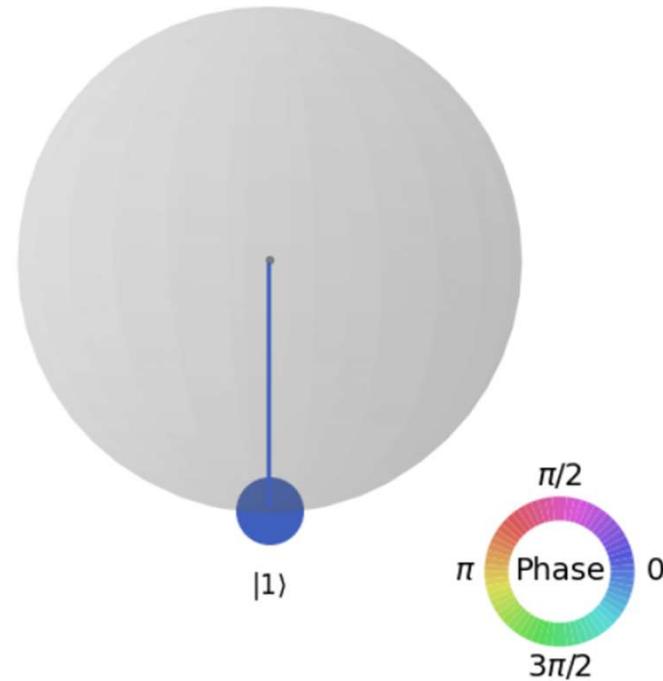
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Representación QUBIT $|1\rangle$

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

[9]:



QUBIT

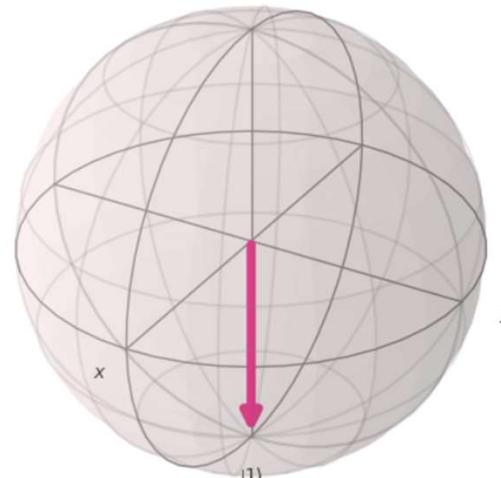
Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

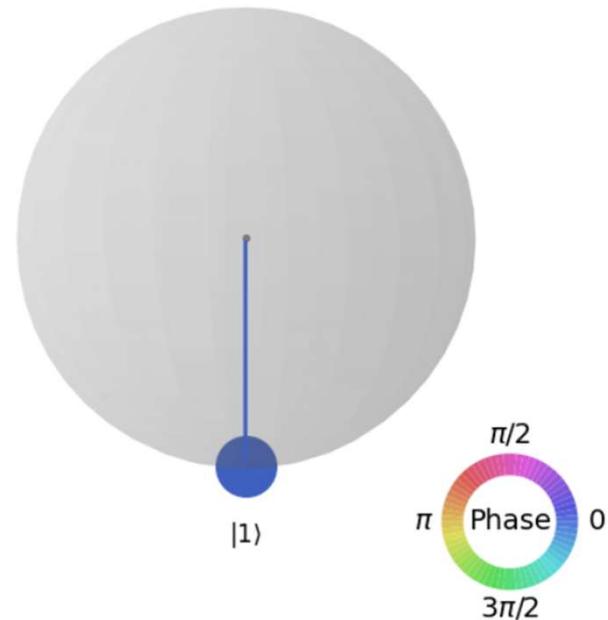
## Representación QUBIT $|1\rangle$

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

qubit 0  
 $|0\rangle$



3



Representación QUBIT  $|+\rangle$ 

```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
qc.h(0)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('bloch')
#####
```

## QUBIT

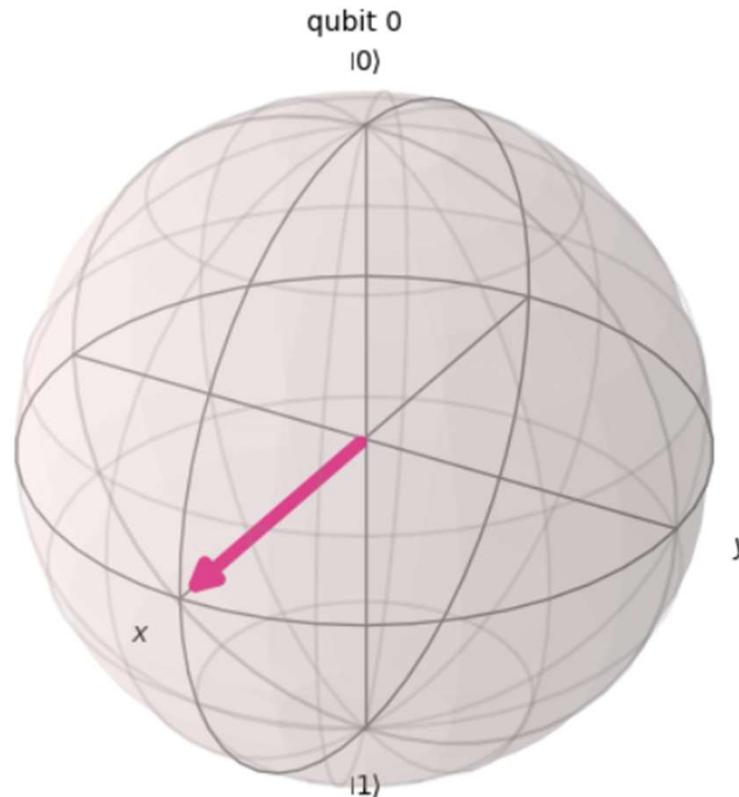
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Representación QUBIT $|+\rangle$

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

[10]:



Representación QUBIT  $|+\rangle$ 

```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
qc.h(0)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('qsphere')
#####
```

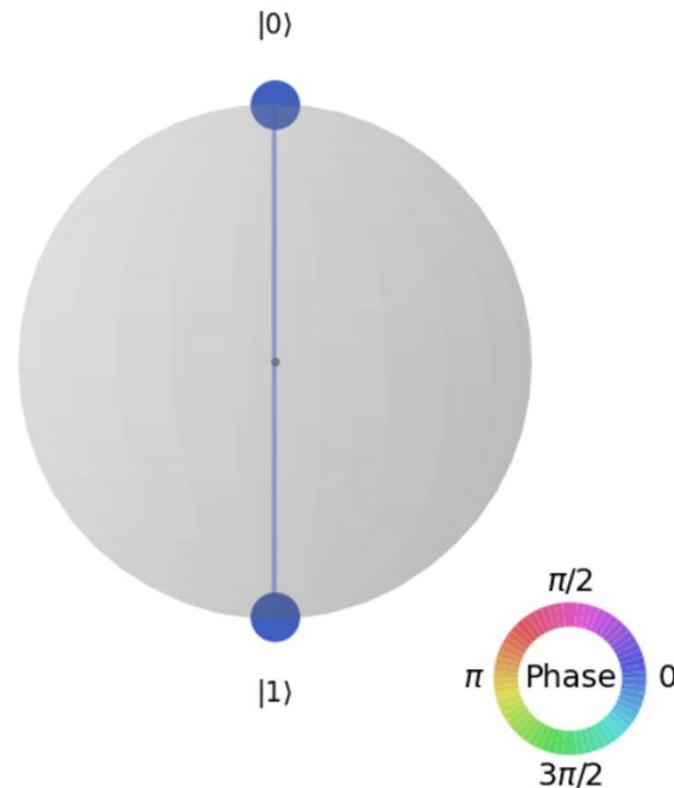
**QUBIT**

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Representación QUBIT $|+\rangle$

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



48



CamScanner

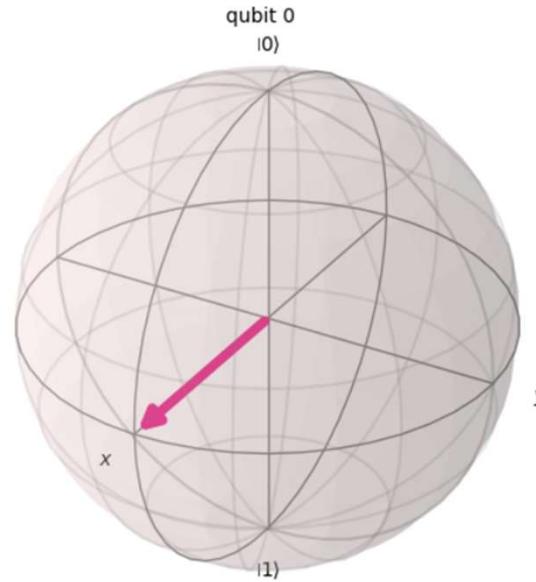
## QUBIT

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

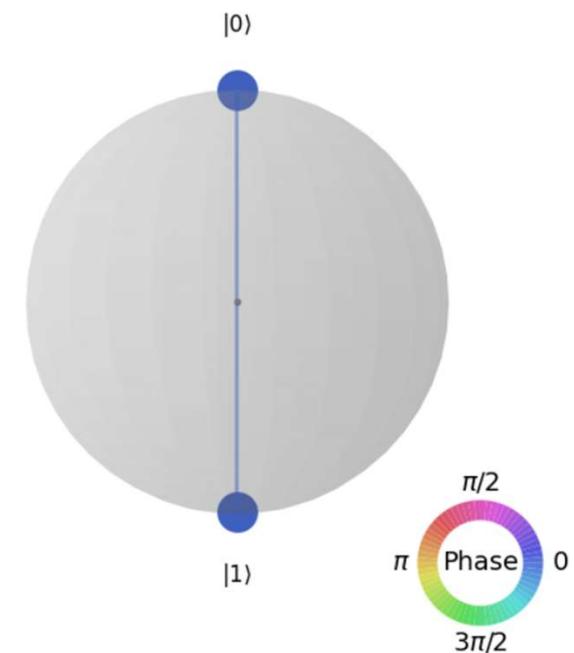
## Representación QUBIT $|+\rangle$

```
[10]: state vector results Statevector([0.70710678+0.j, 0.70710678+0.j], dims=(2,))
```



=

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



## Representación QUBIT |-&gt;

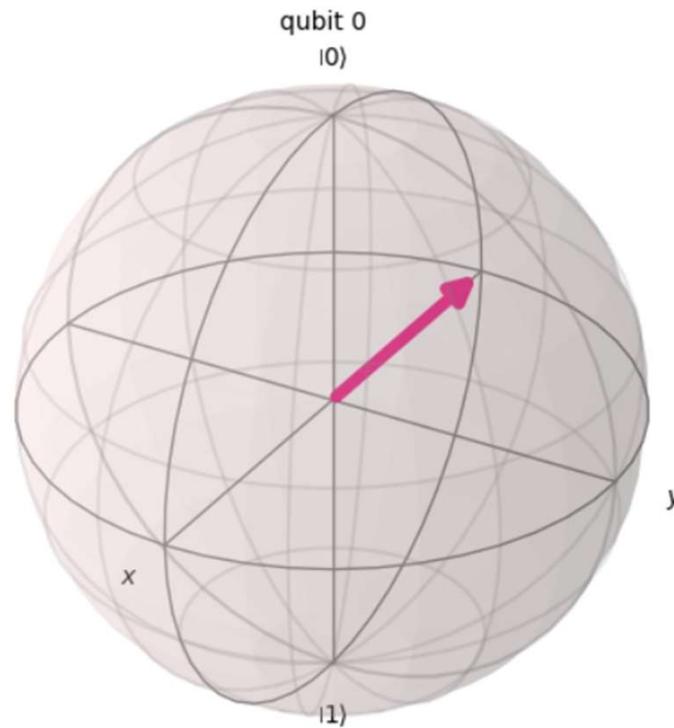
```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
qc.x(0)
qc.h(0)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('bloch')
#####
```

Representación QUBIT  $|-\rangle$ 

```
state vector results Statevector([ 0.70710678+0.j, -0.70710678+0.j],  
    dims=(2,))
```



## Representación QUBIT |-&gt;

```
#####
# CURSO QISKIT 2025
# Esfera de BLOCH

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

qc = QuantumCircuit(1)
qc.x(0)
qc.h(0)
stateVectorResult = Statevector(qc)
print('state vector results', stateVectorResult)
stateVectorResult.draw('qsphere')
#####
```

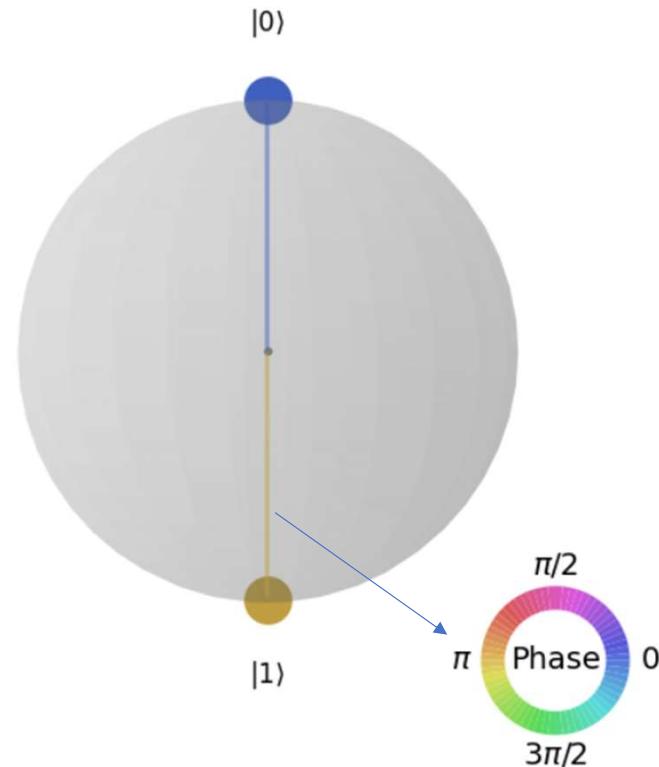
**QUBIT**

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Representación QUBIT $|-\rangle$

```
state vector results Statevector([ 0.70710678+0.j, -0.70710678+0.j],  
dims=(2,))
```



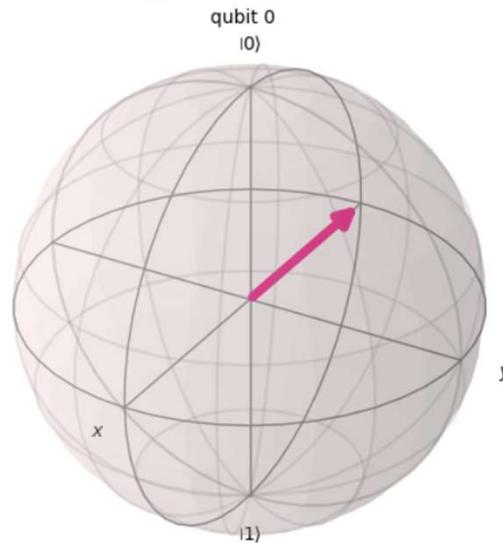
## QUBIT

# Quantum Computing QISKit

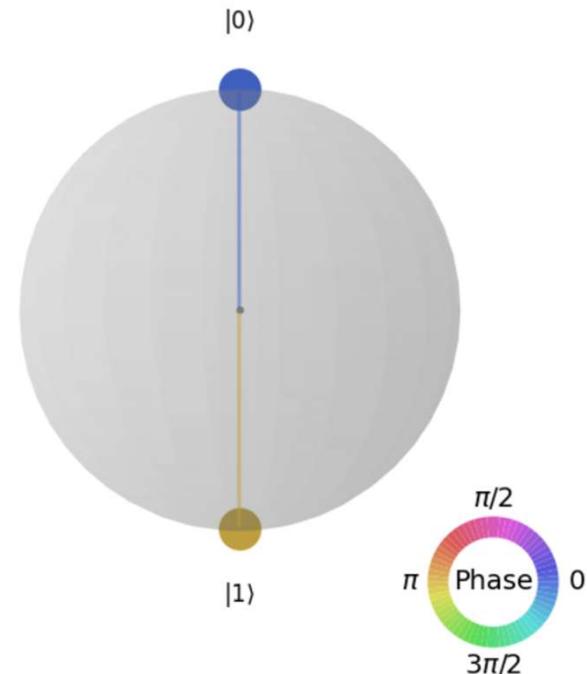
Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Representación QUBIT $|-\rangle$

```
state vector results Statevector([ 0.70710678+0.j, -0.70710678+0.j],  
                               dims=(2,))
```



```
state vector results Statevector([ 0.70710678+0.j, -0.70710678+0.j],  
                               dims=(2,))
```



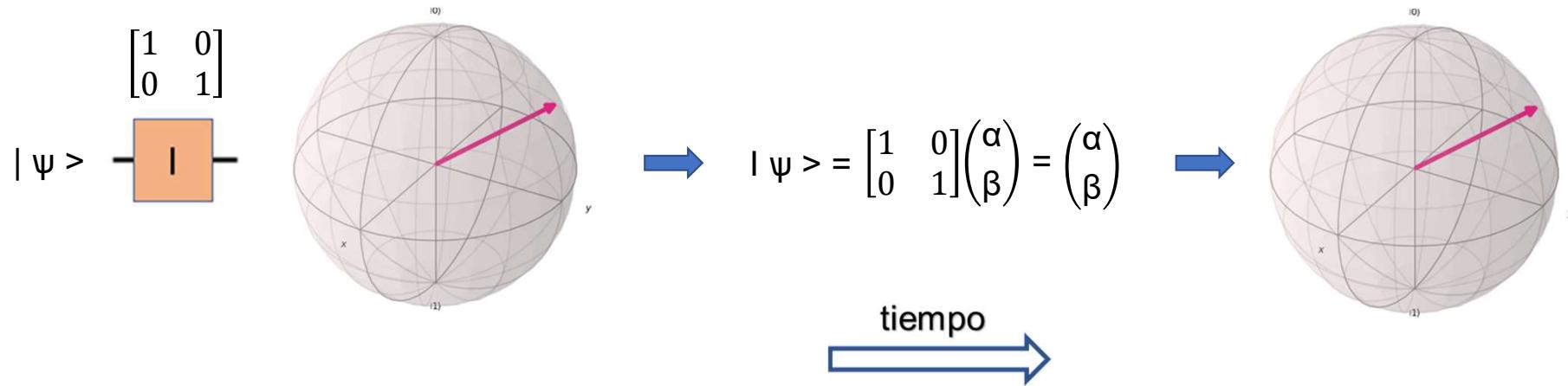
## PAULI GATES

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE I identidad



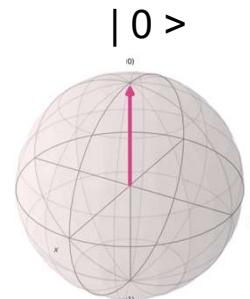
# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

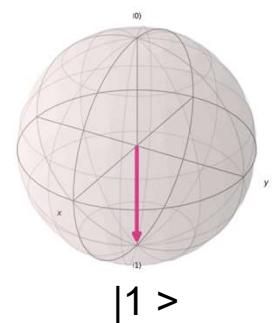
## PAULI GATES

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\psi\rangle - X - |\psi_s\rangle$$
$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



$$|\psi_s\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$
$$\Theta_s = \pi - \Theta_e$$
$$\varphi_s = \varphi_e$$



tiempo  
→

## PAULI GATES GATE X

```
#####
# CURSO QISKIT 2025
# X GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.x(0)
|
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## PAULI GATES GATE X

```
[16]: result
```

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

```
[17]: img
```

```
[17]:
```

$q - \boxed{X} -$

## PAULI GATES

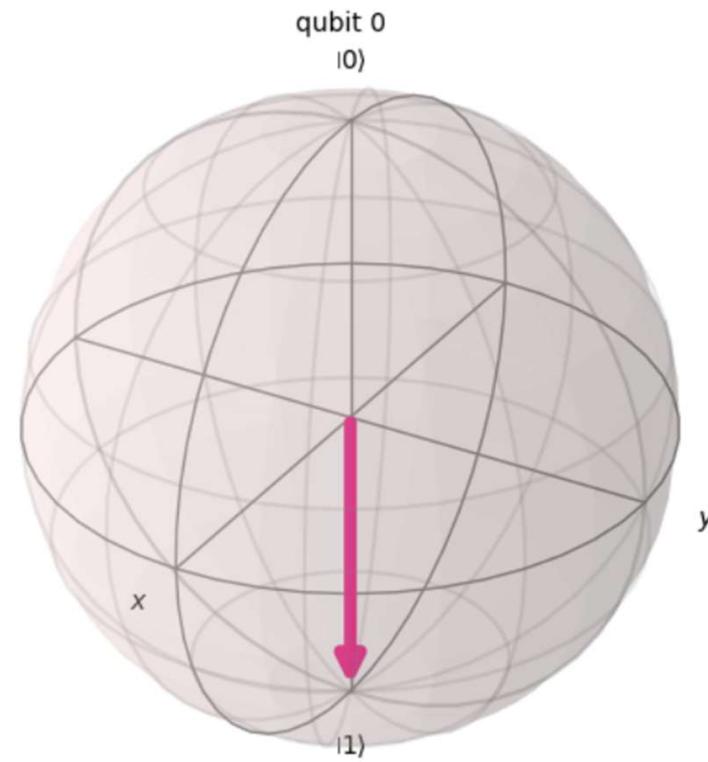
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE X

[18]: bloch\_sphere

[18]:



## PAULI GATES

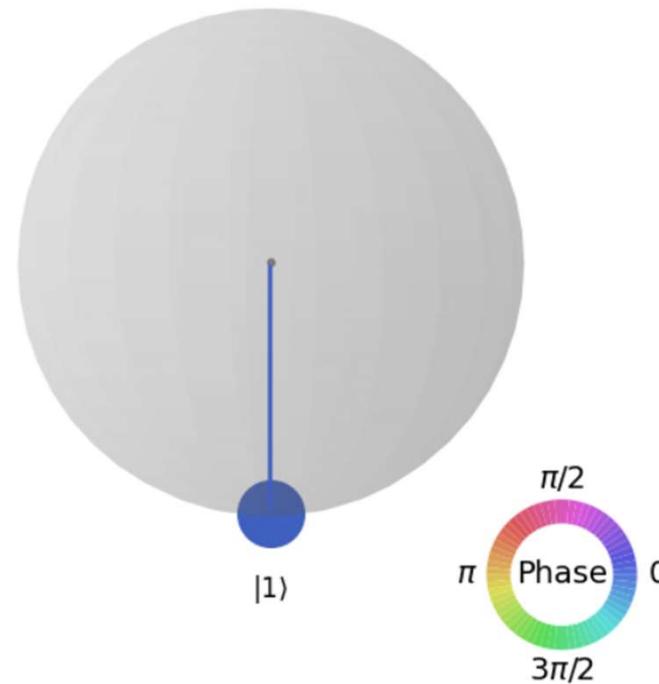
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE X

[19]: qsphere

[19]:



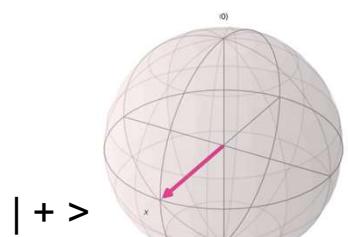
60

## PAULI GATES

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\psi\rangle - \boxed{Y} - |\psi_s\rangle$$

$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$



## Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

### PAULI GATES GATE Y

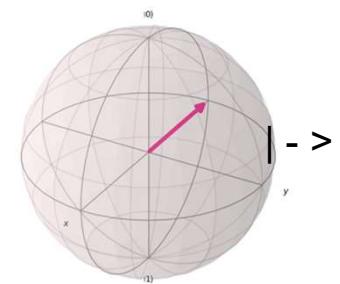
$$\Theta_s = \pi - \Theta_e$$

$$\varphi_s = \pi + \varphi_e$$

$$Fase absoluta no afecta$$

$$|\psi_s\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} -i\beta \\ i\alpha \end{pmatrix} = -i \begin{pmatrix} \beta \\ -\alpha \end{pmatrix}$$

tiempo →



## PAULI GATES GATE Y

```
#####
# CURSO QISKIT 2025
# Y GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.y(0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## PAULI GATES GATE Y

[12]: **result**

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

[13]: **img**

[13]:



## PAULI GATES

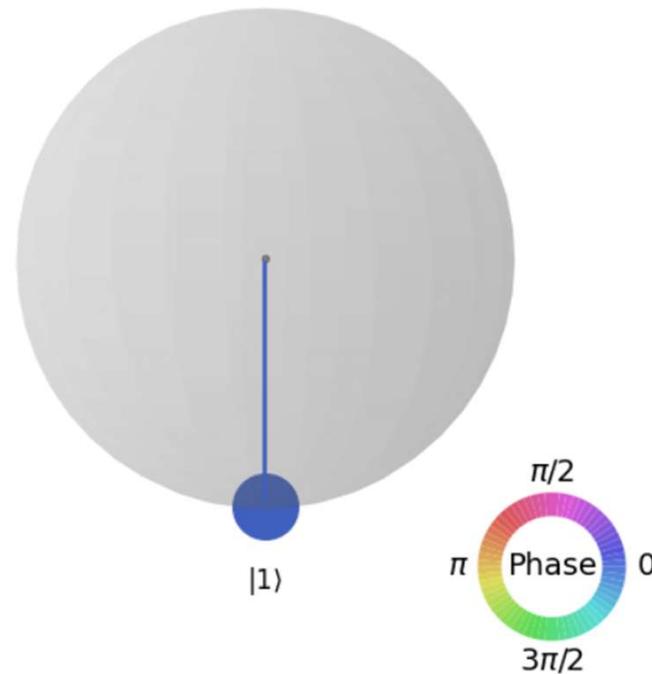
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE Y

[14]: qsphere

[14]:



## PAULI GATES

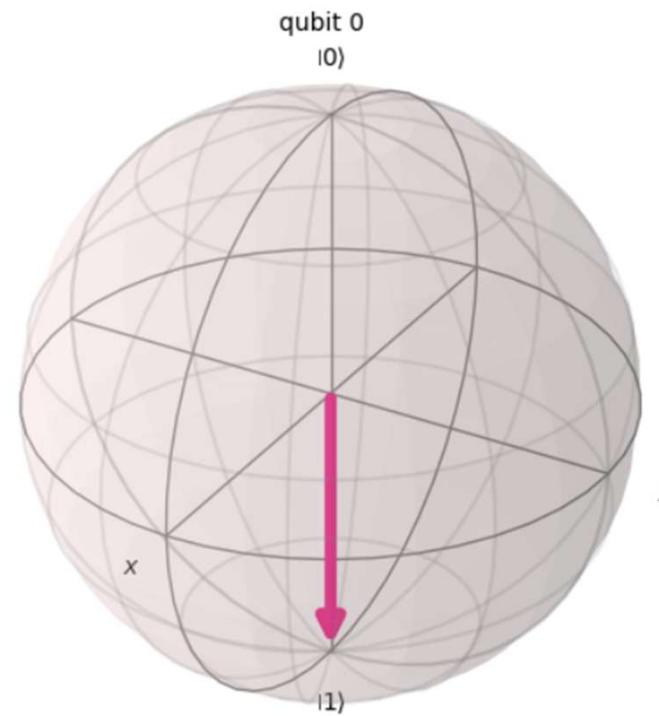
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE Y

[15]: `bloch_sphere`

[15]:



## PAULI GATES

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

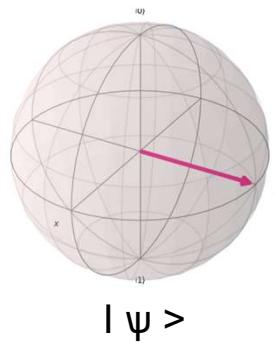
# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE Z

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

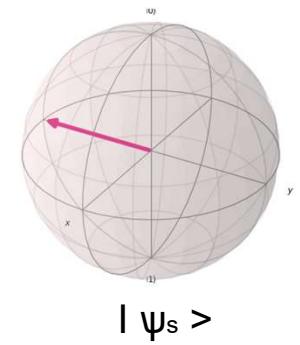
$$|\psi\rangle - \boxed{Z} - |\psi_s\rangle$$



$$|\psi_s\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ -\beta \end{pmatrix}$$

Rota  $\varphi_s = \pi + \varphi_e$

tiempo



## PAULI GATES GATE Z

```
#####
# CURSO QISKIT 2025
# Z GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult

#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere

#-----
qc = QuantumCircuit(1)
qc.x(0) # convierto |0> a |1> para que sea |1> la entrada de la Z Gate
qc.z(0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

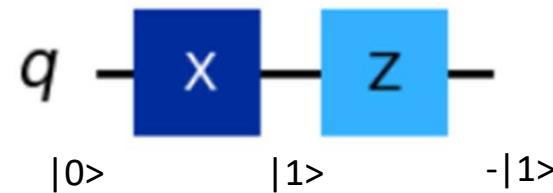
## PAULI GATES GATE Z

```
[17]: result
```

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

```
[18]: img
```

```
[18]:
```



## PAULI GATES

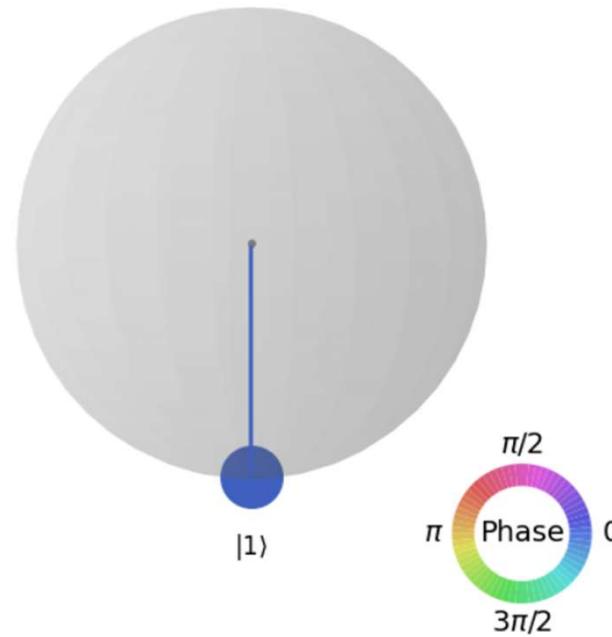
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE Z

[19]: qsphere

[19]:



## PAULI GATES

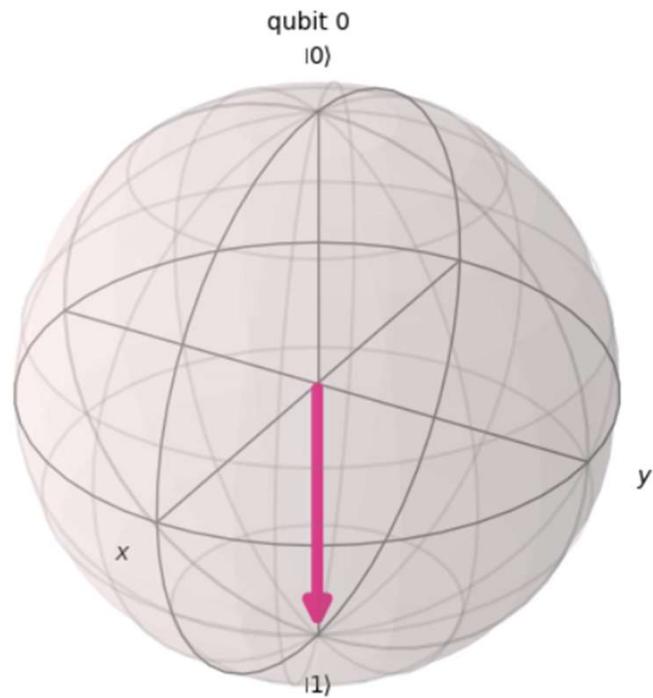
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## PAULI GATES GATE Z

[20]: `bloch_sphere`

[20]:



## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## S GATES



$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ e^{i\pi/2}\beta \end{pmatrix} = \begin{pmatrix} \alpha \\ i\beta \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

$$e^{i\pi/2} = i$$

$$Q[0] = |+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$



$$\text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle + i|1\rangle)$$

## S GATES

```
#####
# CURSO QISKIT 2025
# S GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.h(0)  # pasa |0> a |+> a la entrada de s|
qc.s(0)

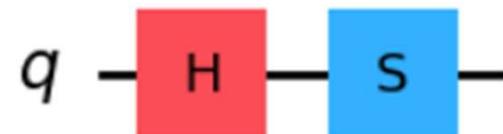
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

```
[27]: result
```

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

```
[28]: img
```

```
[28]:
```



## PHASE GATES

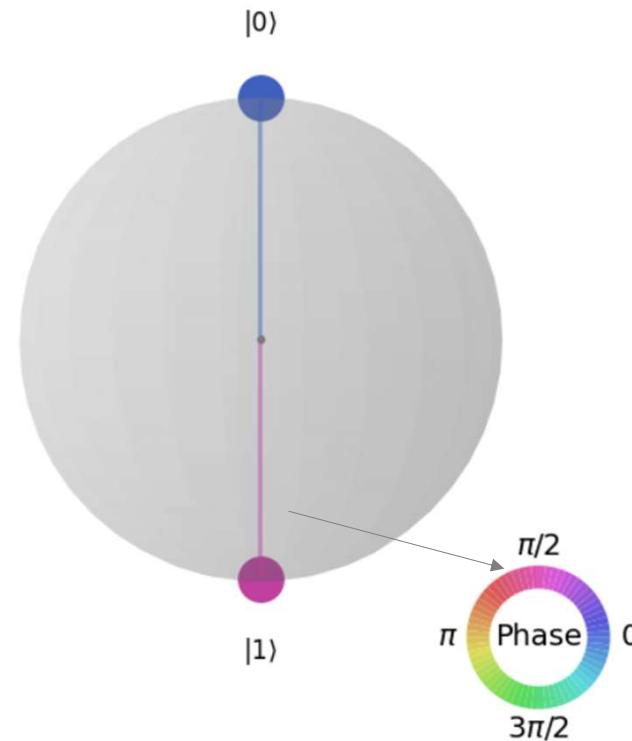
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## S GATES

[29]: qsphere

[29]:



## PHASE GATES

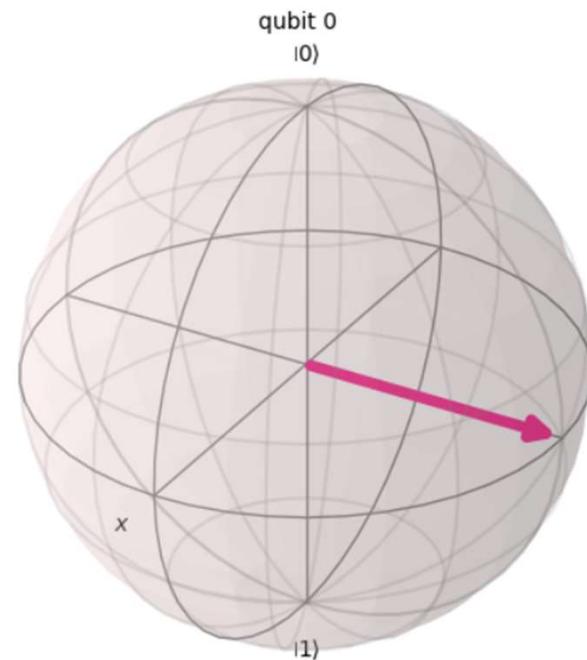
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## S GATES

[30]: `bloch_sphere`

[30]:



$$y \quad \text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle + i |1\rangle)$$

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## S<sup>†</sup> (dagger) GATES

q[0] - S<sup>†</sup> -

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/2} \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ e^{-i\pi/2}\beta \end{pmatrix} = \begin{pmatrix} \alpha \\ -i\beta \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$$

$$e^{-i\pi/2} = -i$$

$$Q[0] = |+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$



$$\text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle - i|1\rangle)$$

## S† (dagger) GATES

```
#####
# CURSO QISKIT 2025
# S dagger GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.h(0) # pasa |0> a |+> a la entrada de s
qc.sdg(0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

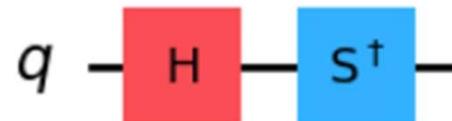
S<sup>†</sup> (dagger) GATES

[32]: result

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

[33]: img

[33]:



## PHASE GATES

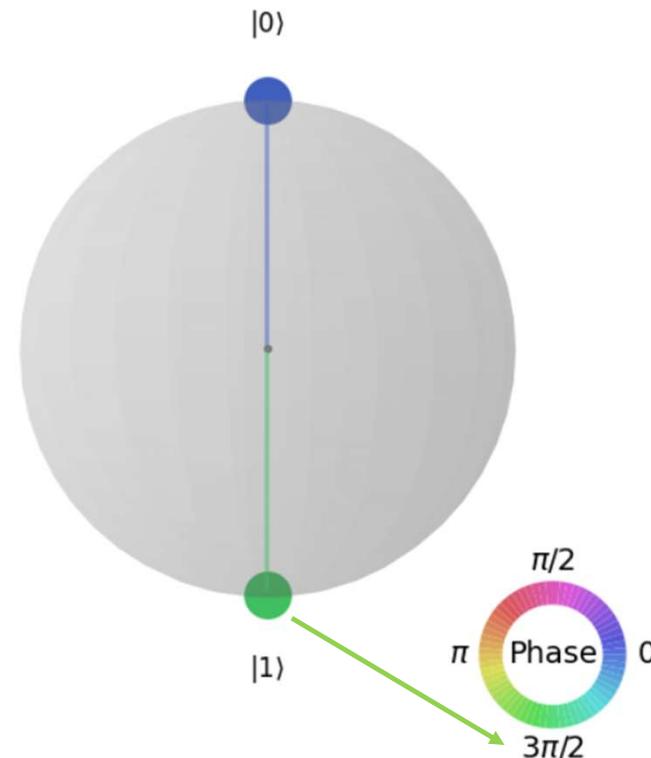
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## S† (dagger) GATES

[34]: qsphere

[34]:



$$\text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle - i |1\rangle)$$

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## S† (dagger) GATES

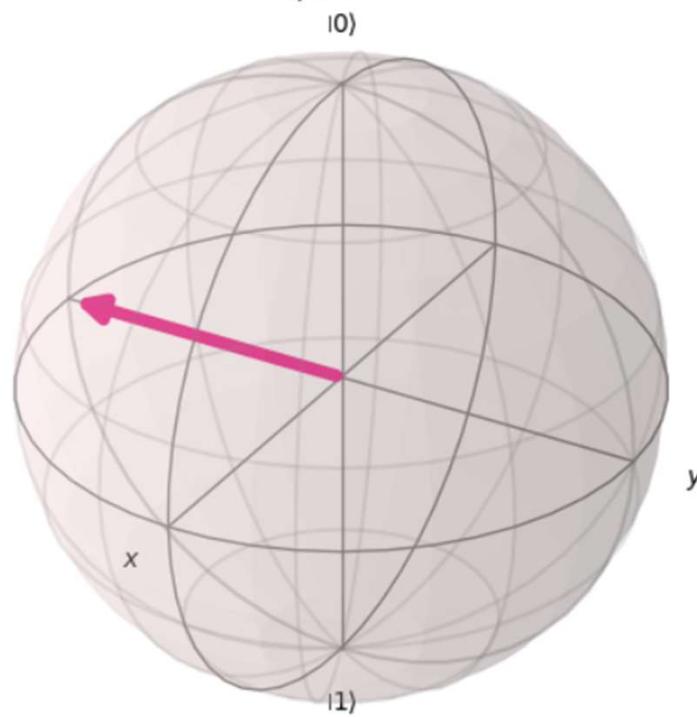
[35]: bloch\_sphere

[35]:

qubit 0

|0>

$$\text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle - i |1\rangle)$$



## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## T GATES



$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ e^{i\pi/4}\beta \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

$$e^{i\pi/2} = i$$

$$Q[0] = |+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$



$$\text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\pi/4}|1\rangle)$$

## T GATES

```
#####
# CURSO QISKIT 2025
# T GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.h(0) # pasa |0> a |+> a la entrada de s
qc.t(0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

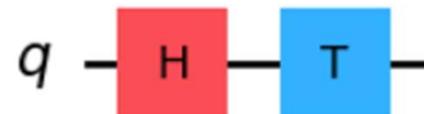
## T GATES

[37]: result

```
Statevector([0.70710678+0.j , 0.5 +0.5j],  
           dims=(2,))
```

[38]: img

[38]:



## PHASE GATES

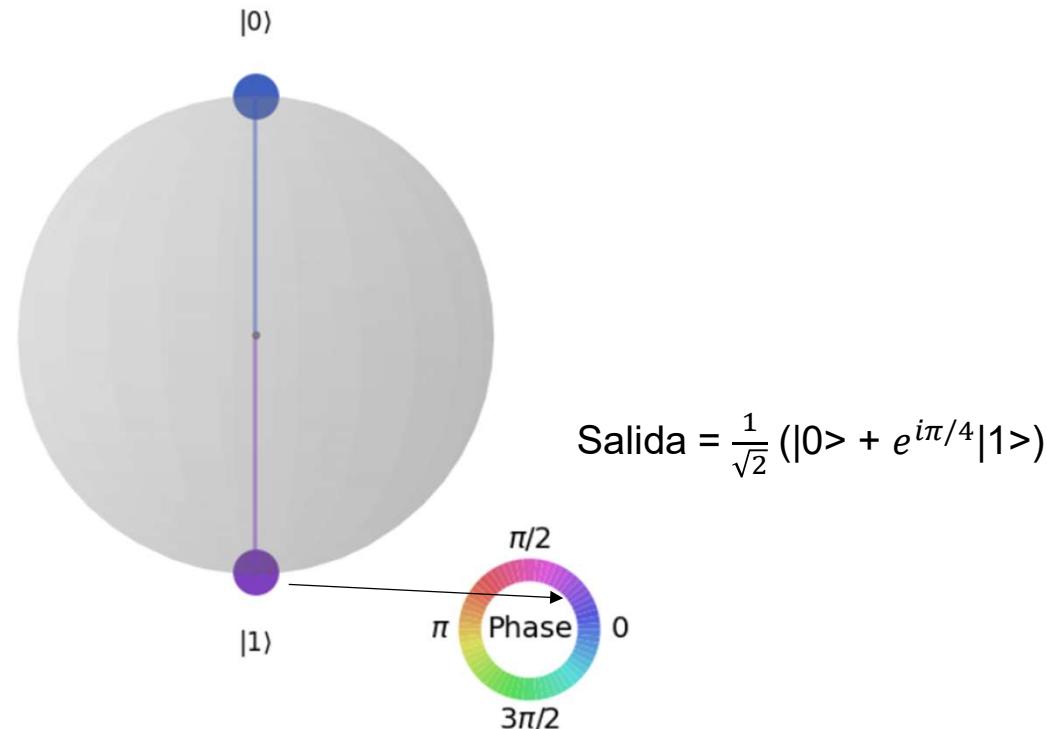
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## T GATES

[39]: qsphere

[39]:



## PHASE GATES

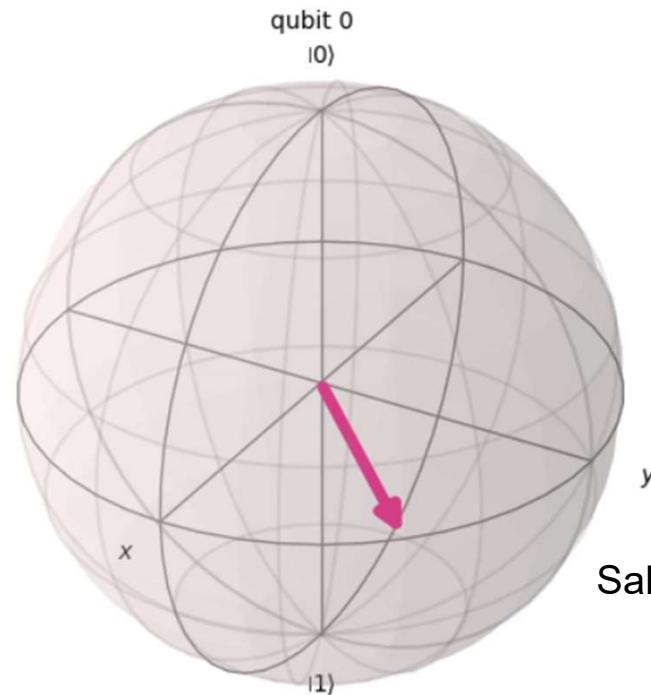
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## T GATES

[40]: `bloch_sphere`

[40]:



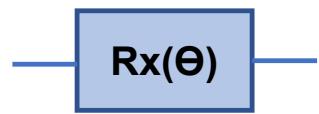
$$\text{Salida} = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\pi/4}|1\rangle)$$

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## R GATES



$$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

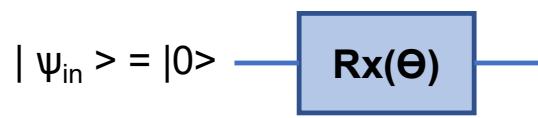


$$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$



$$\begin{bmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

## Rx GATES



$$|\psi_{out}\rangle = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \cos\left(\frac{\theta}{2}\right) |0\rangle - i \sin\left(\frac{\theta}{2}\right) |1\rangle$$

Tomando  $\Theta = \pi/6 = 30$

$$|\psi_{out}\rangle = \cos\left(\frac{\pi}{12}\right) |0\rangle - i \sin\left(\frac{\pi}{12}\right) |1\rangle = 0,9659|0\rangle - i 0,2588|1\rangle$$

## Rx GATES

```
#####
# CURSO QISKIT 2025
# Rx GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

from math import pi

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult

#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere

#-----
qc = QuantumCircuit(1)
qc.rx(pi/6,0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Rx GATES

[2]: result

```
statevector([0.96592583+0.j      , 0.        -0.25881905j],  
           dims=(2,))
```

[3]: img

[3]:



## PHASE GATES

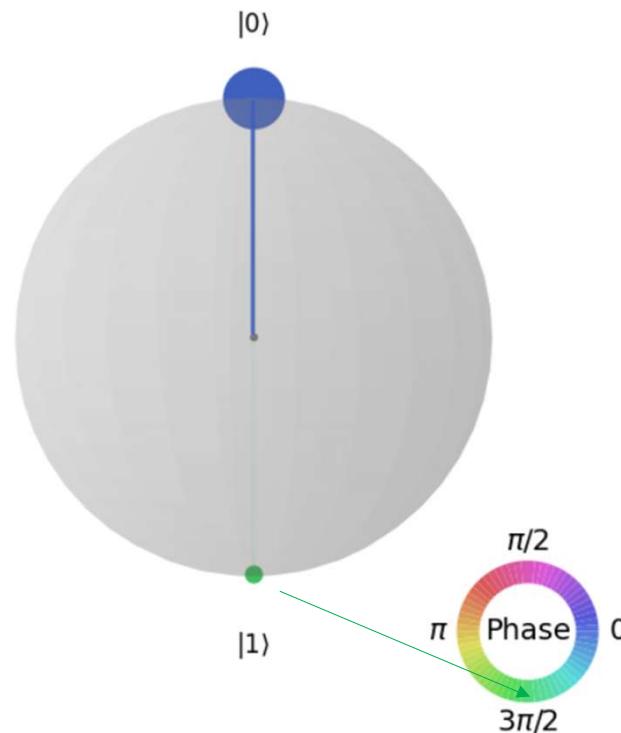
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Rx GATES

[4]: qsphere

[4]:



90

## PHASE GATES

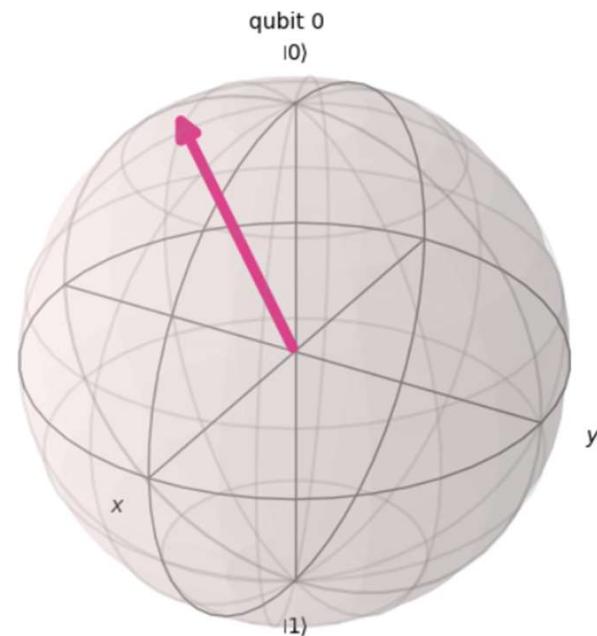
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

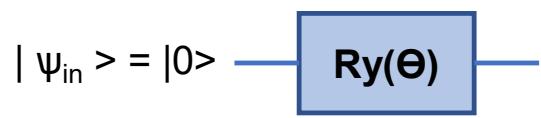
## R GATES

[5]: `bloch_sphere`

[5]:



## Ry GATES



$$|\psi_{out}\rangle = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)|1\rangle$$

Tomando  $\Theta = \pi/6 = 30$

$$|\psi_{out}\rangle = \cos\left(\frac{\pi}{12}\right)|0\rangle + \sin\left(\frac{\pi}{12}\right)|1\rangle = 0,9659|0\rangle + 0,2588|1\rangle$$

## Ry GATES

```
#####
# CURSO QISKIT 2025
# Ry GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *

from math import pi

#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.ry(pi/6,0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Ry GATES

[23]: result

```
Statevector([0.96592583+0.j, 0.25881905+0.j],  
           dims=(2,))
```

[24]: img

[24]:



## PHASE GATES

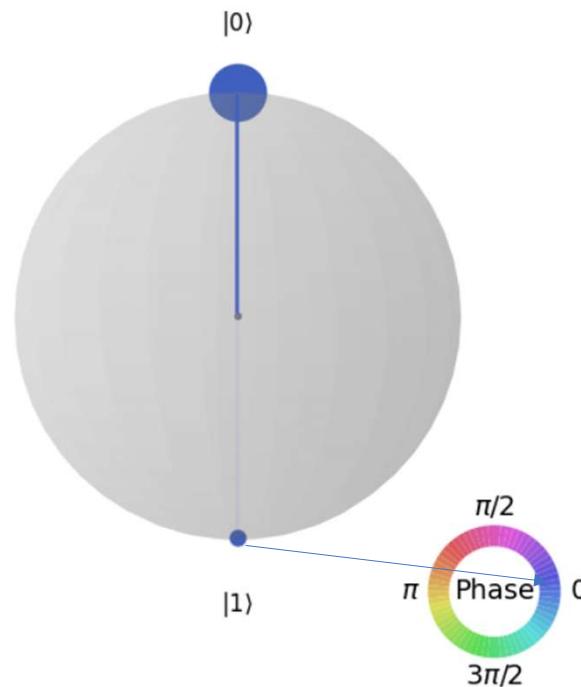
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Ry GATES

[26]: qsphere

[26]:



## PHASE GATES

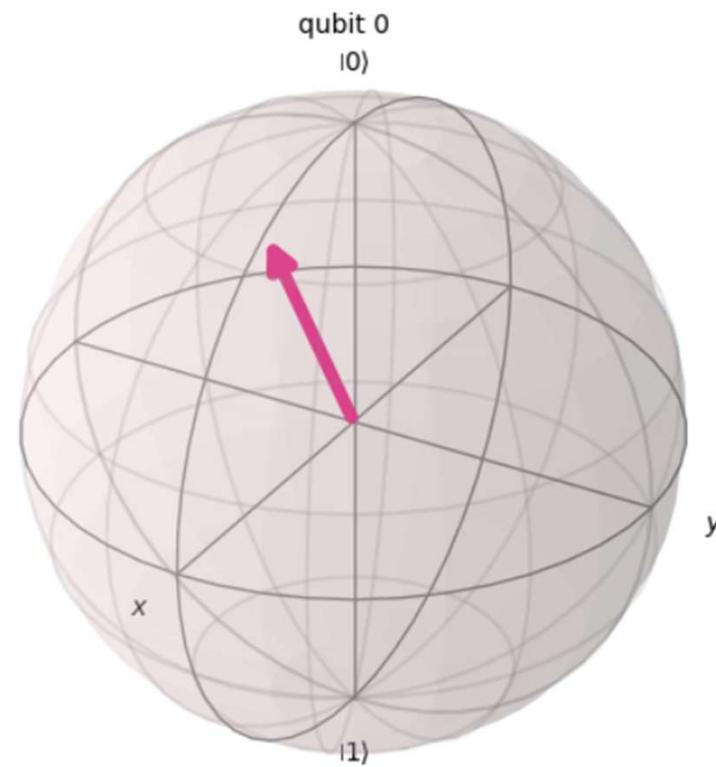
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Ry GATES

[25]: `bloch_sphere`

[25]:



## Rz GATES

$$|\psi_{in}\rangle = |0\rangle \xrightarrow{\text{Rz}(\Theta)} |\psi_{out}\rangle = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = e^{-i\theta/2} |0\rangle = \\ = (\cos(\theta/2) - i \sin(\theta/2)) |0\rangle$$

Tomando  $\Theta = \pi/6 = 30^\circ$

$$\left\{ \begin{array}{l} \cos(\theta/2) = \cos(15) = 0,9659 \\ \sin(\theta/2) = \sin(15) = 0,2588 \end{array} \right. \quad |\psi_{out}\rangle = (0.9659 - i0,2588) |0\rangle$$

## Rz GATES

```
#####
# CURSO QISKIT 2025
# Rz GATES

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.rz(pi/6,0)

result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## PHASE GATES

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

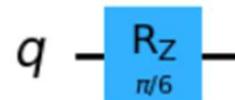
## Rz GATES

[28]: result

```
Statevector([0.96592583-0.25881905j, 0.           +0.j        ],  
          dims=(2,))
```

[29]: img

[29]:



## PHASE GATES

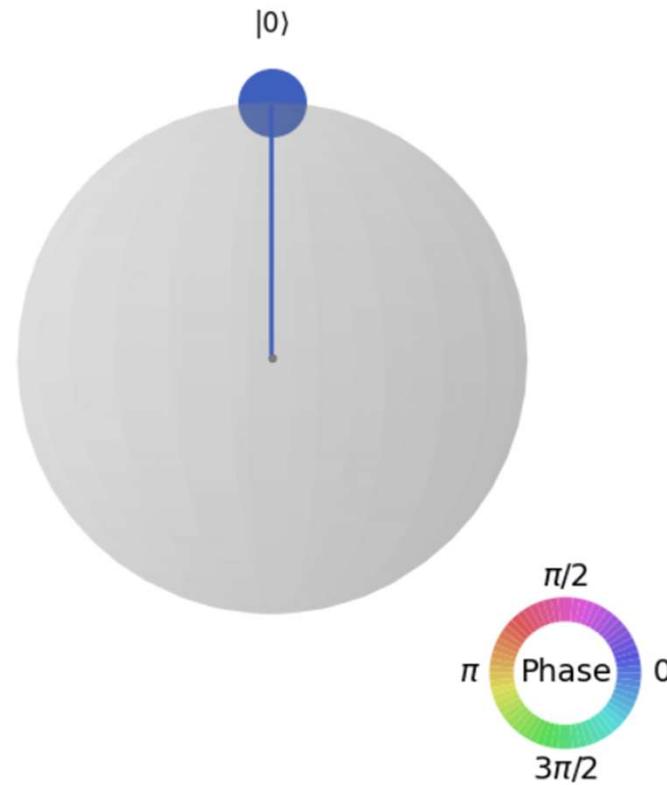
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Rz GATES

[30]: qsphere

[30]:



100

## PHASE GATES

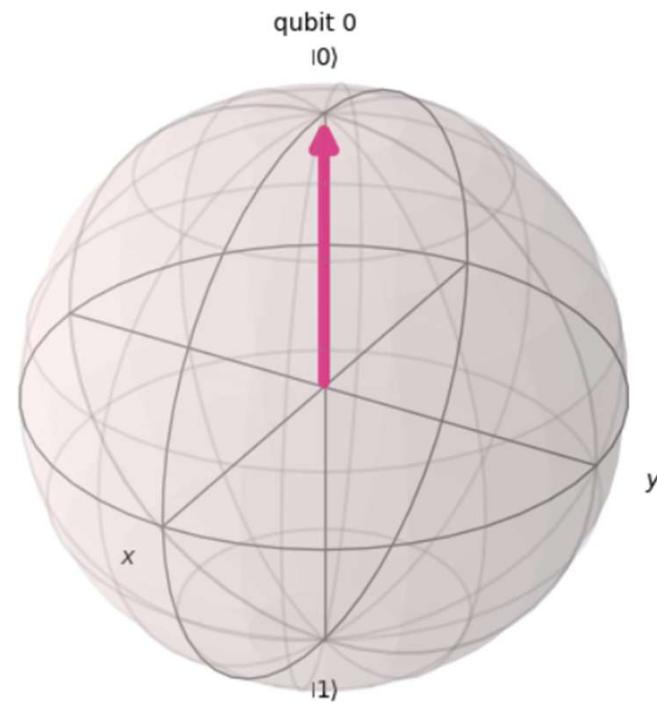
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Rz GATES

[31]: `bloch_sphere`

[31]:



UNIVERSAL U GATE

Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

U GATE

$$U(\Theta, \phi, \lambda) \quad \begin{bmatrix} \cos\left(\frac{\Theta}{2}\right) & -e^{-i\lambda}\sin\left(\frac{\Theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\Theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\Theta}{2}\right) \end{bmatrix}$$

$$|\psi_{in}\rangle = |0\rangle \quad \text{U}(\Theta, \phi, \lambda) \quad |\psi_{out}\rangle = \begin{bmatrix} \cos\left(\frac{\Theta}{2}\right) & -e^{-i\lambda}\sin\left(\frac{\Theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\Theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\Theta}{2}\right) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \cos\left(\frac{\Theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\Theta}{2}\right)|1\rangle$$

Tomando  $\Theta = \phi = \lambda = \pi/2 = 90^\circ$

$$|\psi_{out}\rangle = \cos\left(\frac{\pi/2}{2}\right)|0\rangle + e^{i\pi/2}\sin\left(\frac{\pi/2}{2}\right)|1\rangle = 0,707|0\rangle + i0,707|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$$

## UNIVERSAL U GATE

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## U GATE

```
#####
# CURSO QISKIT 2025
# UNIVERSAL GATE

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(1)
qc.u(pi/2, pi/2, pi/2, 0)
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## UNIVERSAL U GATE

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## U GATE

[3]: result

```
Statevector([7.07106781e-01+0.j      , 4.32978028e-17+0.70710678j],  
           dims=(2,))
```

[5]: img

[5]:

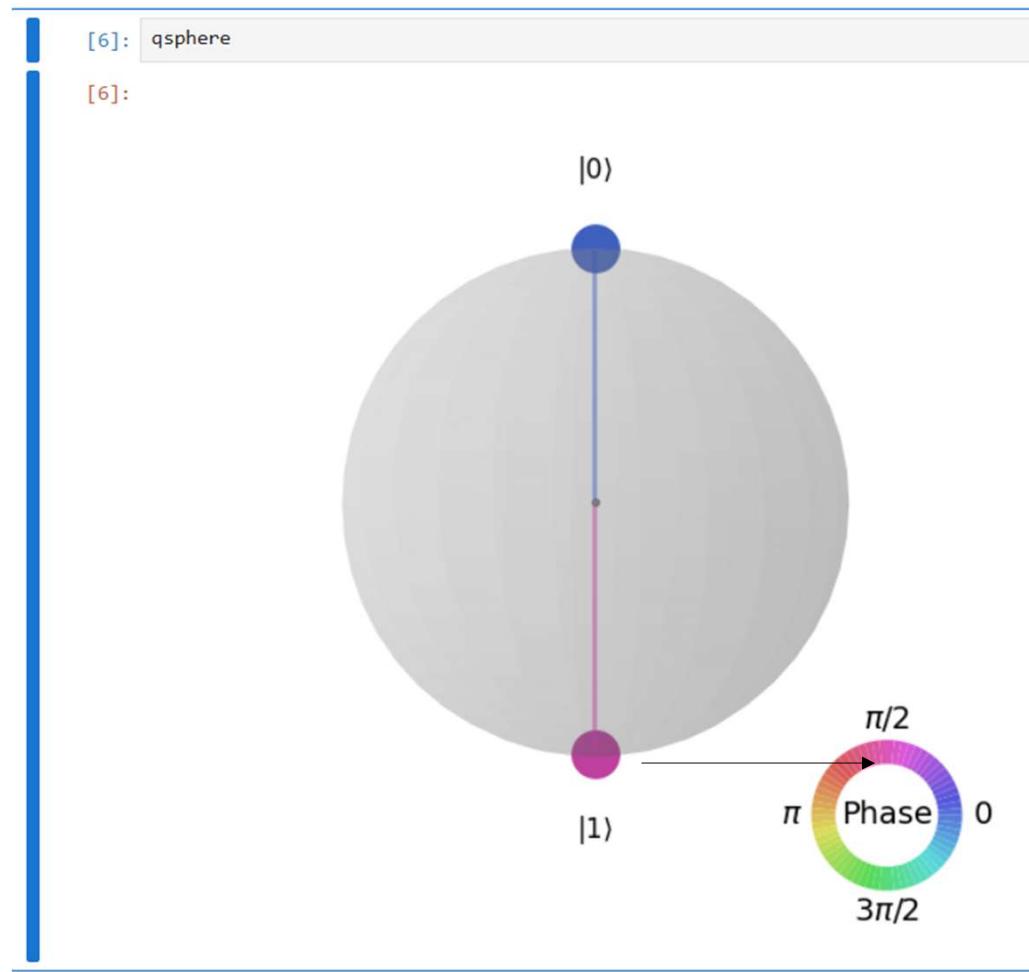


## UNIVERSAL U GATE

# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## U GATE



105

UNIVERSAL U GATE

# Quantum Computing QISKIT

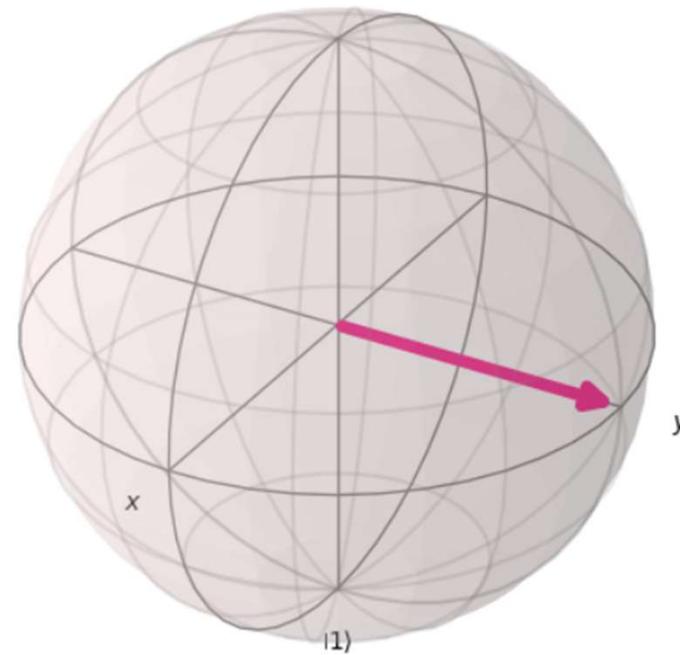
Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

U GATE

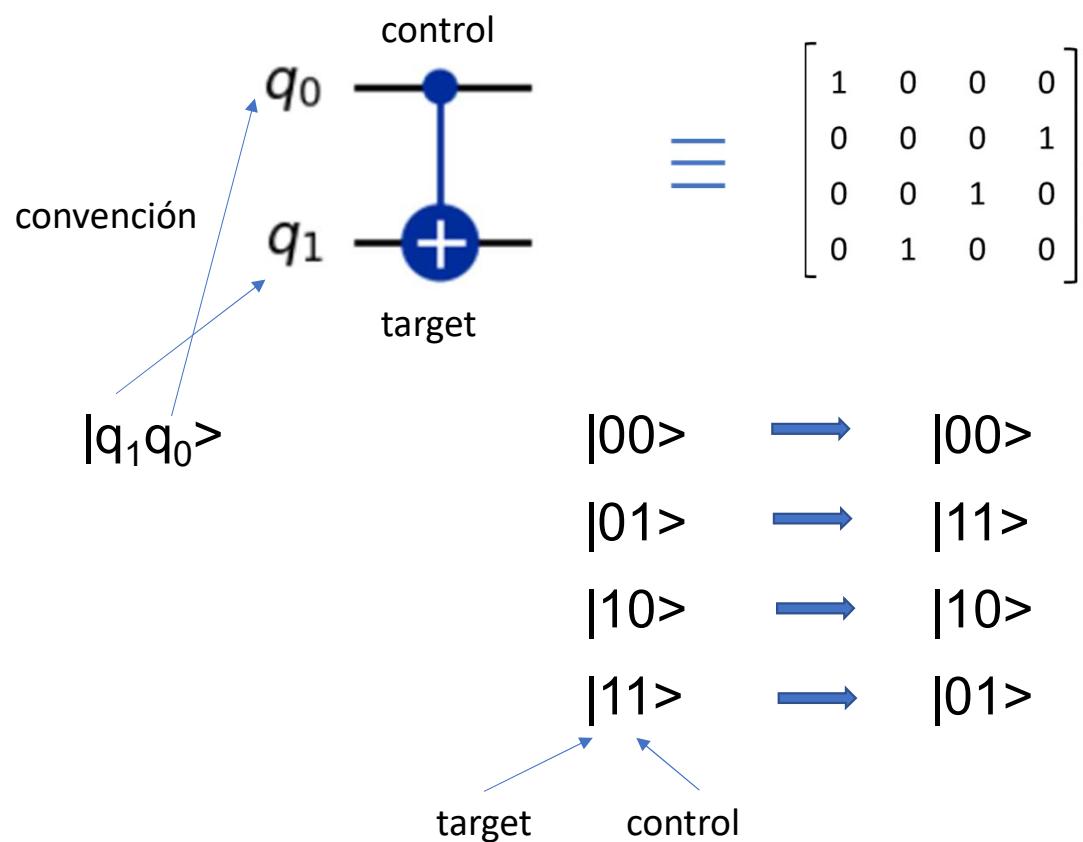
[7]: `bloch_sphere`

[7]:

qubit 0  
 $|0\rangle$



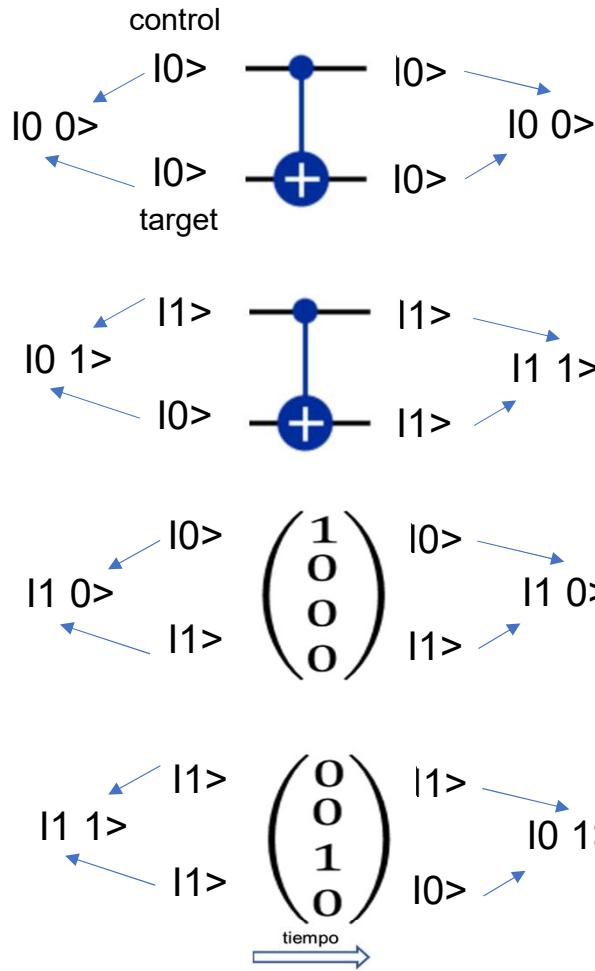
C - NOT



# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Multi Qubit Gates



C - NOT

$$|\Psi_{\text{out}}\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |0\ 0\rangle$$

$$|\Psi_{\text{out}}\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = |1\ 1\rangle$$

$$|\Psi_{\text{out}}\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |1\ 0\rangle$$

$$|\Psi_{\text{out}}\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |0\ 1\rangle$$

108



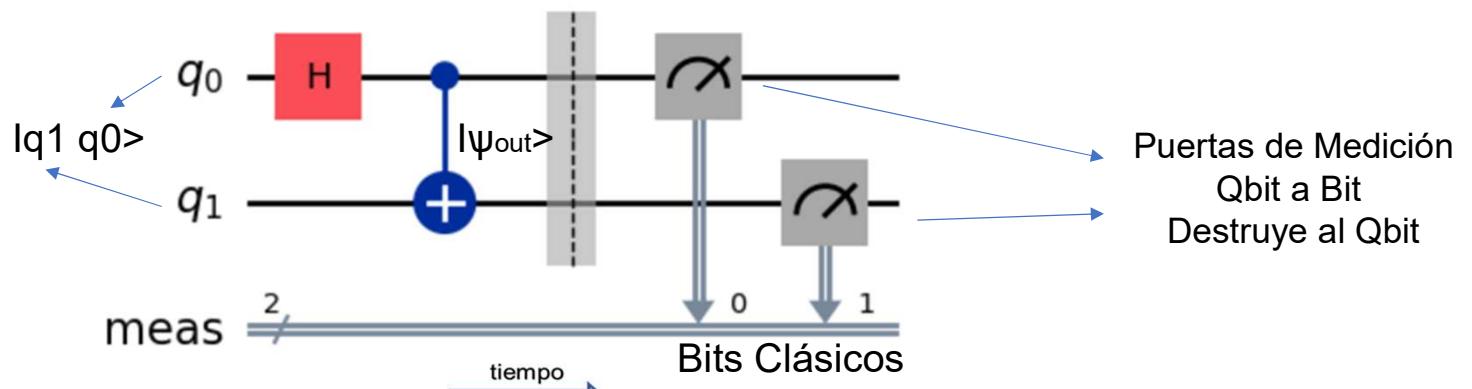
CamScanner

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Multi Qubit Gates

C - NOT



$$|\Psi_{out}\rangle = (C\text{-NOT}) \times (I \otimes H) |q_1 q_0\rangle$$

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

109

## C - NOT

Tomando  $|\psi_{\text{in}}\rangle = |00\rangle$

$$|\psi_{\text{out}}\rangle = U |00\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$$

$$|\psi_{\text{out}}\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$$

## C - NOT

```
#####
# CURSO QISKIT 2025
# Multi Qubit CNOT GATE

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)

    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0,1)
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

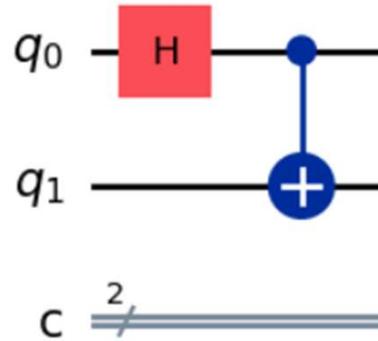
## C - NOT

[13]: result

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

[14]: img

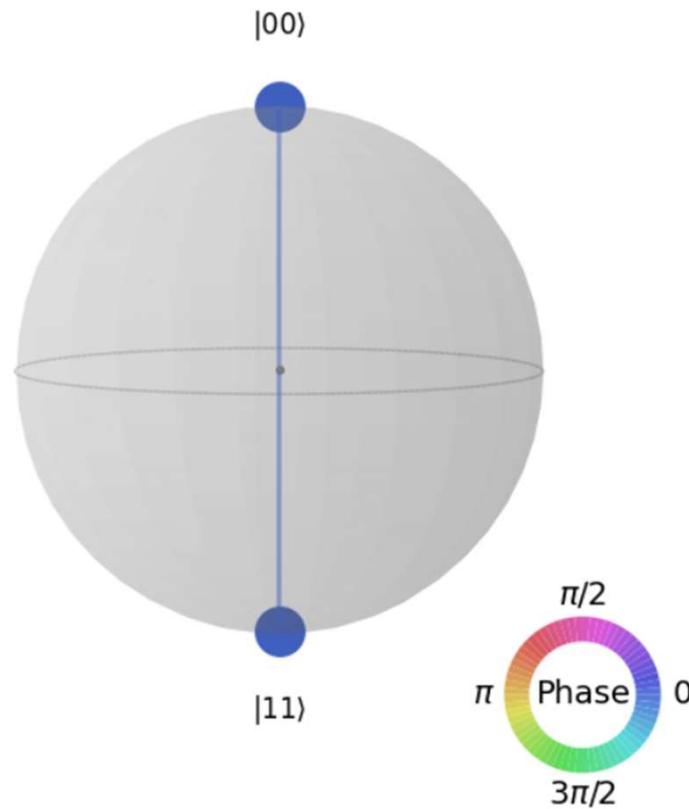
[14]:



## C - NOT

[15]: qsphere

[15]:



## C – NOT histograma

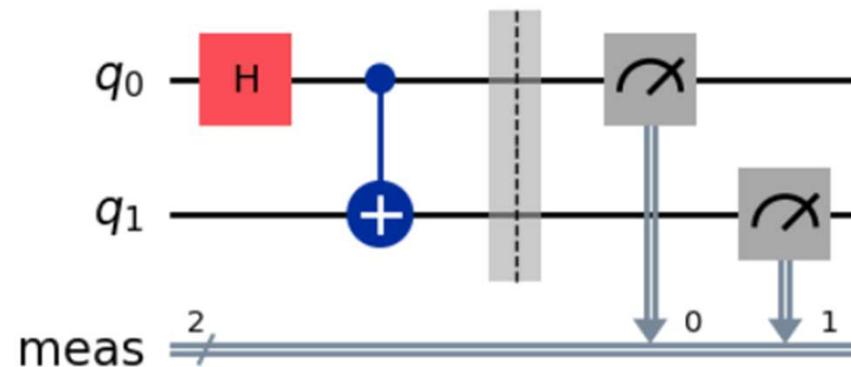
```
#####
# CURSO QISKIT 2025
# Multi Qubit CNOT GATE con medidor

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from qiskit import transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from math import pi
-----
# Will run the circuit
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def exe_circuit(quantum_circuit):
    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Run
    simulator = AerSimulator()
    compiled_circuit = transpile(quantum_circuit, simulator)
    sim_result = simulator.run(compiled_circuit, shots= 1000, memory=True).result()
    counts = sim_result.get_counts()
    pl = plot_histogram(counts)
    #Return the results
    return circuit_diagram, counts, pl
-----
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0,1)
qc.measure_all()
circ_diag, counts_, pl_ =  exe_circuit(qc)
print(counts_)
#####
```

## C – NOT histograma

```
[2]: circ_diag
```

```
[2]:
```



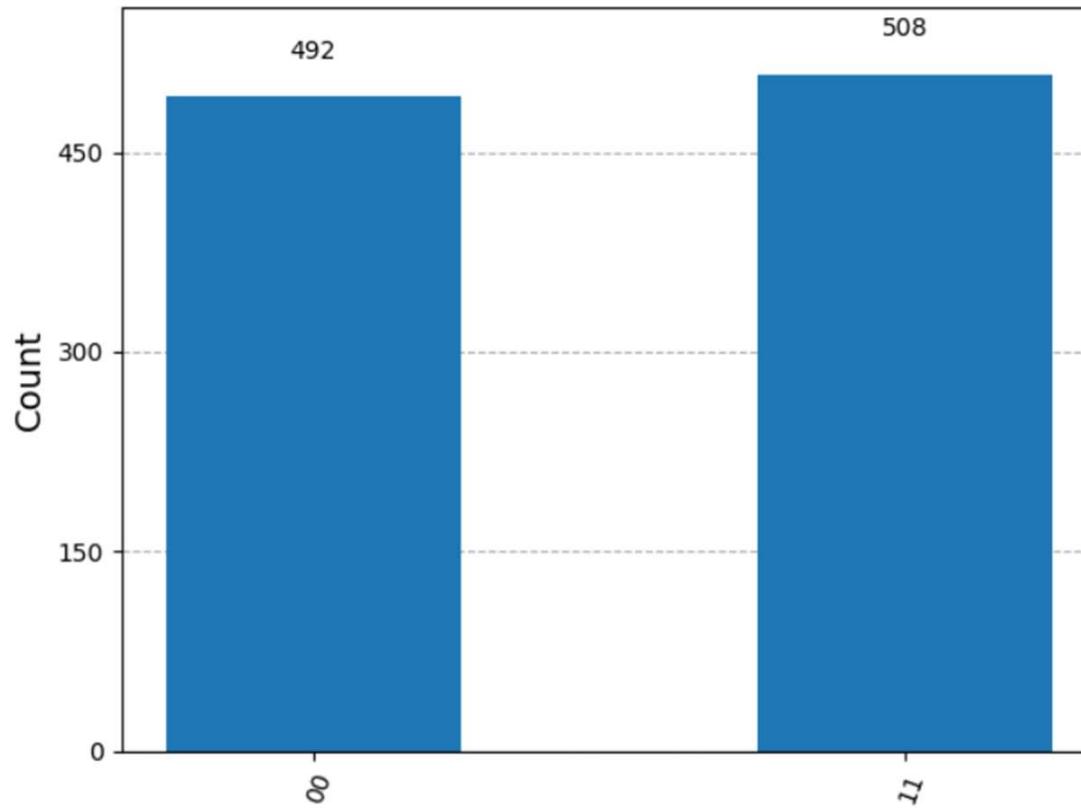
```
[5]: print(counts_)
```

```
{'00': 492, '11': 508}
```

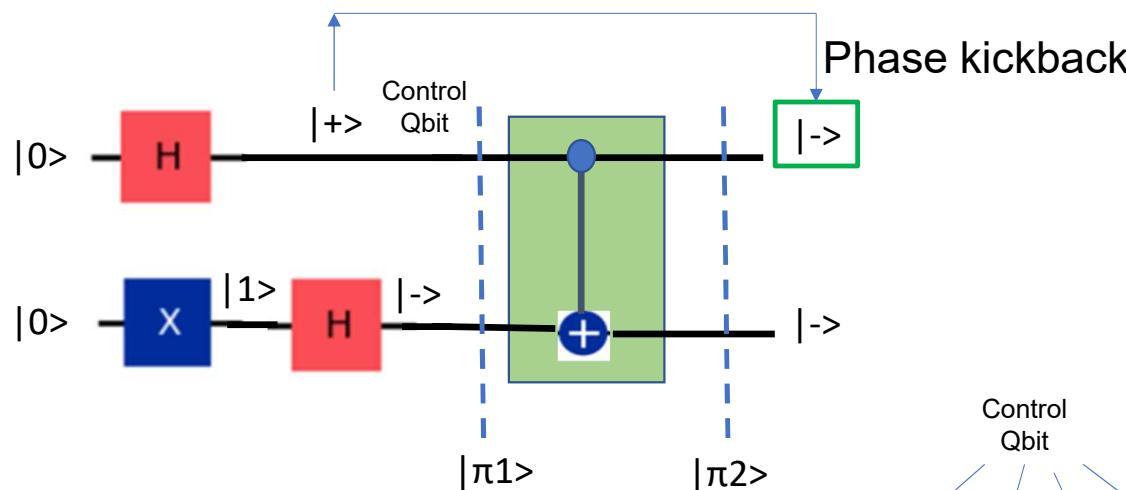
## C – NOT histograma

[4]: p1\_

[4]:



Phase Kickback (algoritmo Deutsch lo utiliza)



$$|\pi_1\rangle = |-\rangle \otimes |+\rangle = \frac{1}{2} (|0\rangle - |1\rangle) \otimes (|0\rangle + |1\rangle) = \frac{1}{2} (|00\rangle + |01\rangle - |10\rangle - |11\rangle)$$

$$|\pi_2\rangle = |-\rangle \otimes |+\rangle = \frac{1}{2} (|00\rangle + |11\rangle - |10\rangle - |01\rangle) = \frac{1}{2} (|0\rangle - |1\rangle) \otimes (|0\rangle - |1\rangle) = (|-\rangle \otimes |-\rangle)$$

$$|\pi_2\rangle = |\text{OUT}\rangle = (|-\rangle \otimes |-\rangle) = |--\rangle = \frac{1}{2} [ |00\rangle + |11\rangle - |01\rangle - |10\rangle ]$$

## Phase Kickback

```
#####
# CURSO QISKIT 2025
# Multi Qubit PHASE KICK BACK
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)
    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(2,2)
qc.h(0)
qc.x(1)
qc.h(1)
qc.barrier()
qc.cx(0,1)
qc.barrier()
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

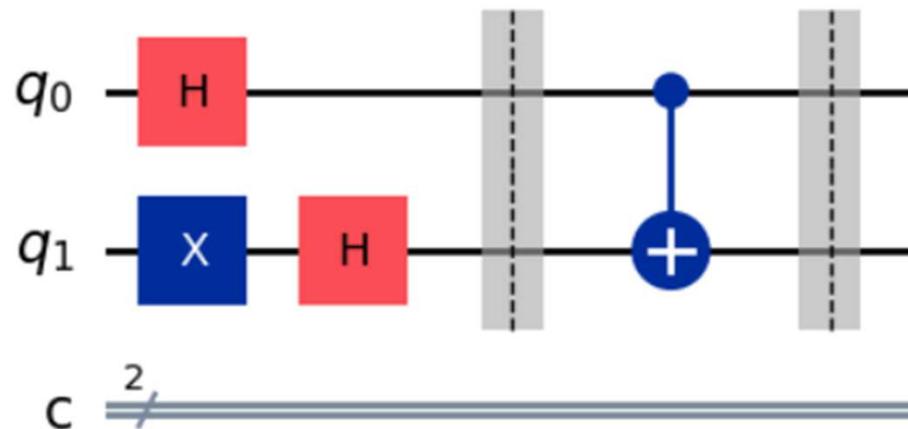
## Phase Kickback

```
[2]: result
```

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

```
[3]: img
```

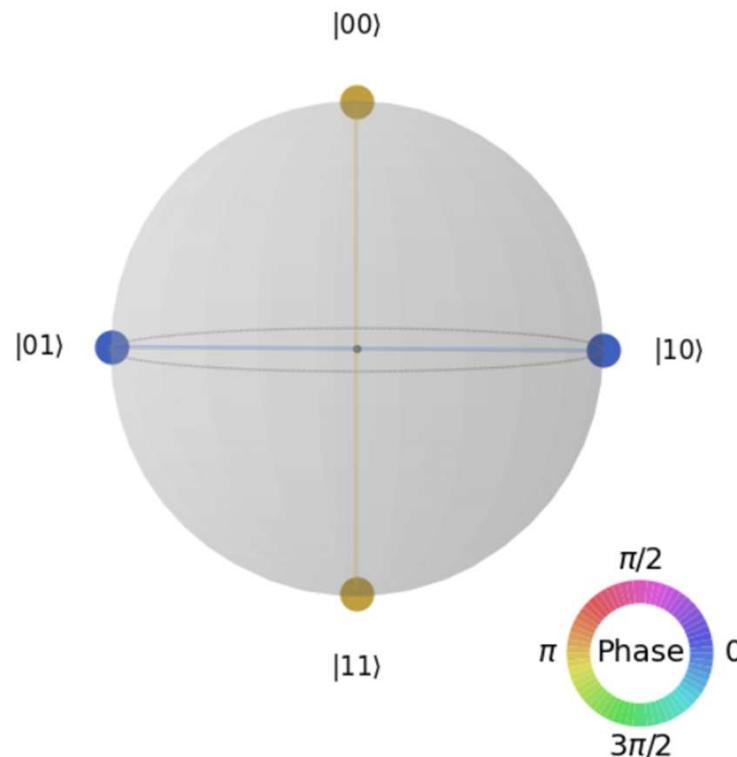
```
[3]:
```



## Phase Kickback

[4]: qsphere

[4]:



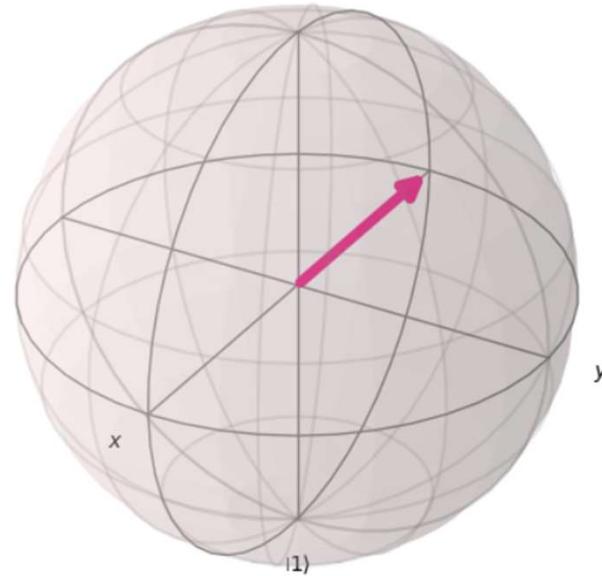
## Phase Kickback

```
[5]: bloch_sphere
```

```
[5]:
```

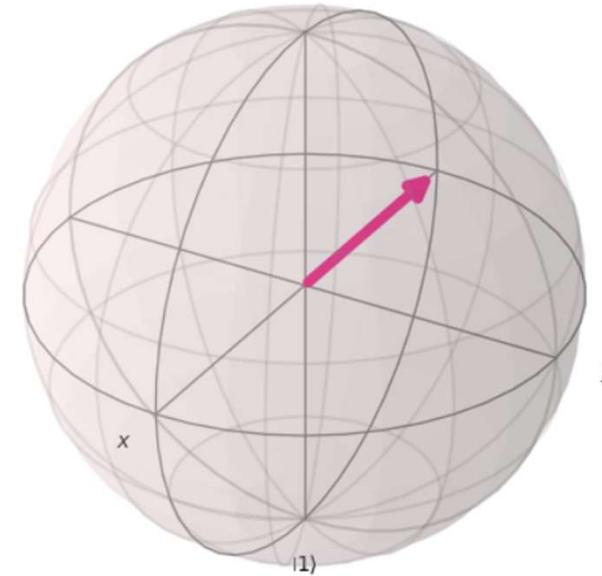
qubit 0

$|0\rangle$

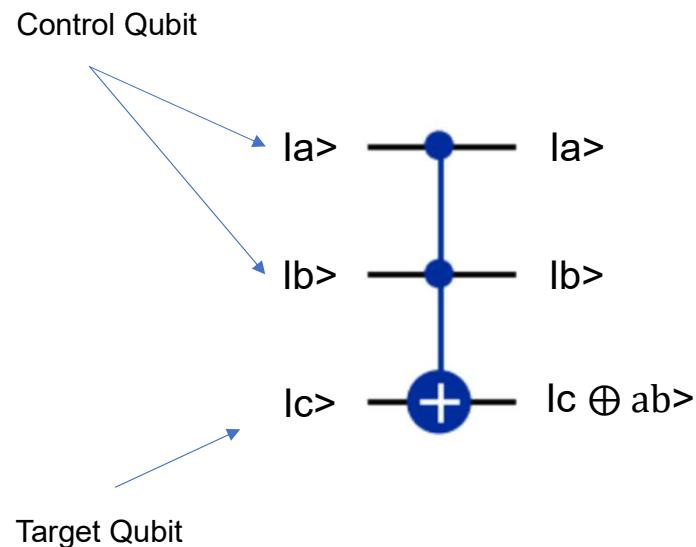


qubit 1

$|0\rangle$



## Toffoli Gate



$$\equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Dimensión 8x8 – 3 qbits

## Toffoli Gate

```
#####
# CURSO QISKIT 2025
# Multi Qubit TOFFOLI
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)
    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
-----
qc = QuantumCircuit(3)
qc.x(range(2))
qc.ccx(0,1,2)
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## Toffoli Gate

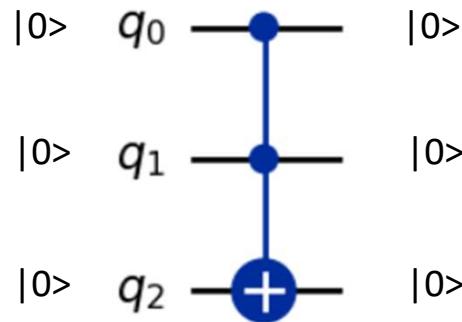
[12]: result

```
Statevector([1.+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))
```

[13]: img

[13]:

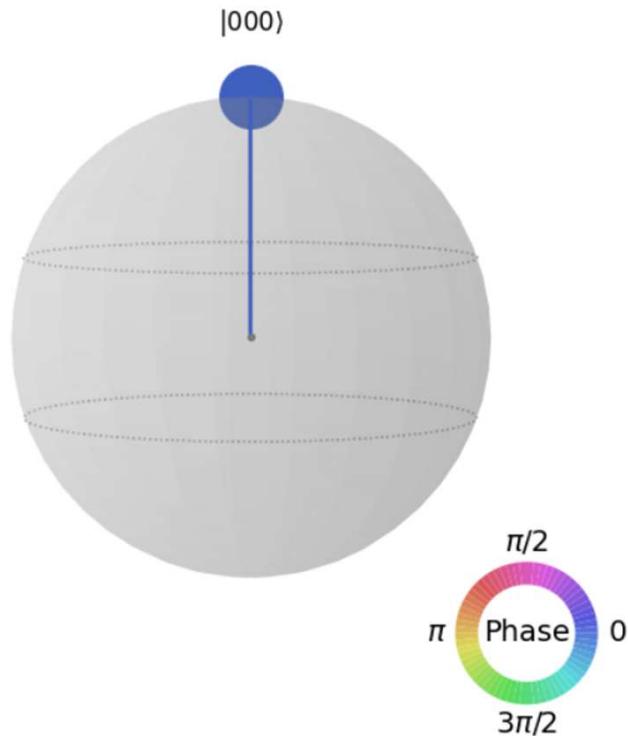
$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



## Toffoli Gate

[14]: qsphere

[14]:



## Multi Qubit Gates

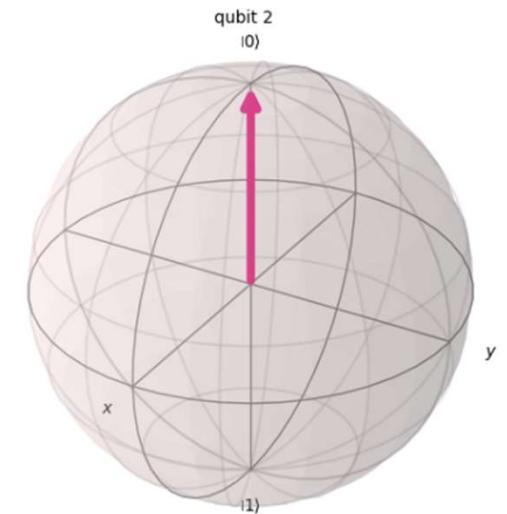
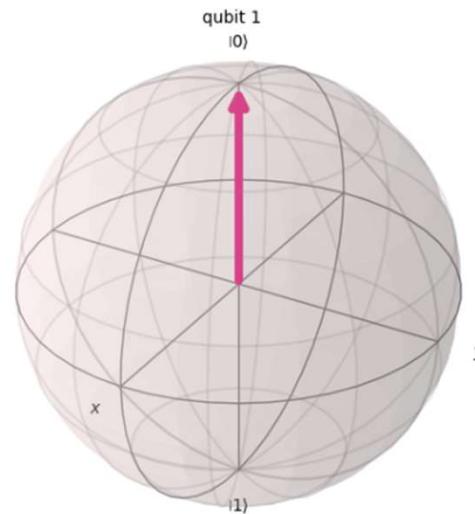
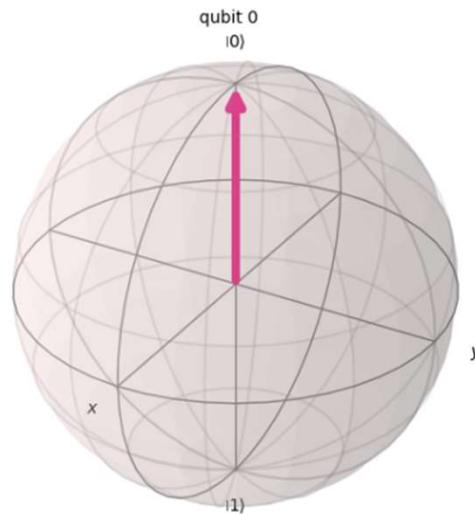
# Quantum Computing QISKIT

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Toffoli Gate

[15]: bloch\_sphere

[15]:



## Toffoli Gate

```
#####
# CURSO QISKIT 2025
# Multi Qubit TOFFOLI
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)
    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#
qc = QuantumCircuit(3)
qc.x(range(2))
qc.ccx(0,1,2)
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## Toffoli Gate

[17]: result

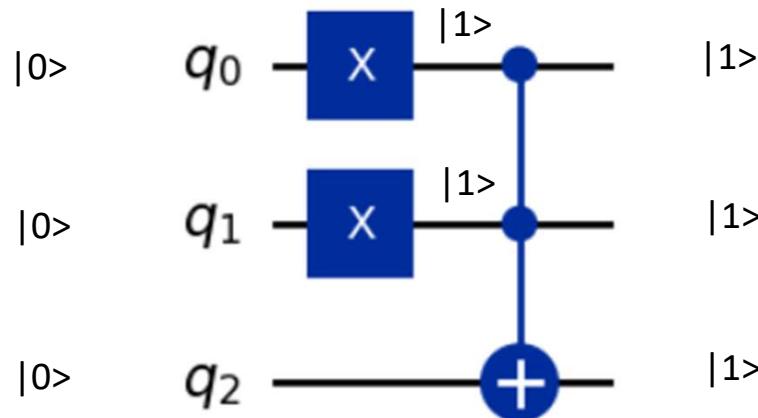
```
Statevector([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,  
           1.+0.j],  
           dims=(2, 2, 2))
```



$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

[18]: img

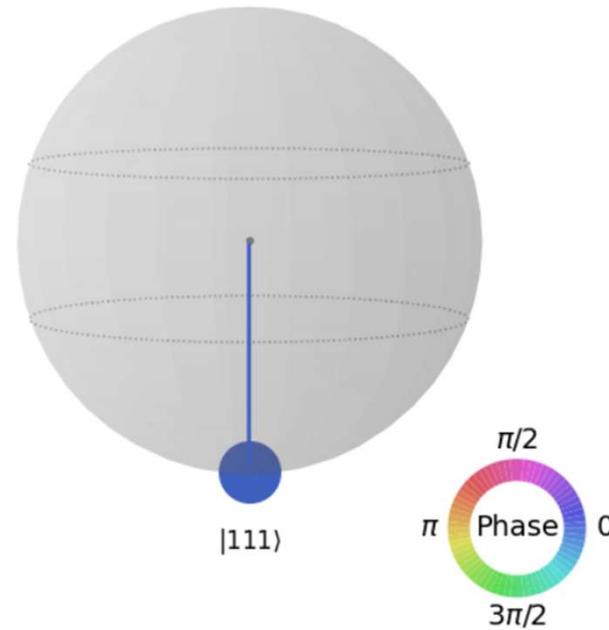
[18]:



## Toffoli Gate

[19]: qsphere

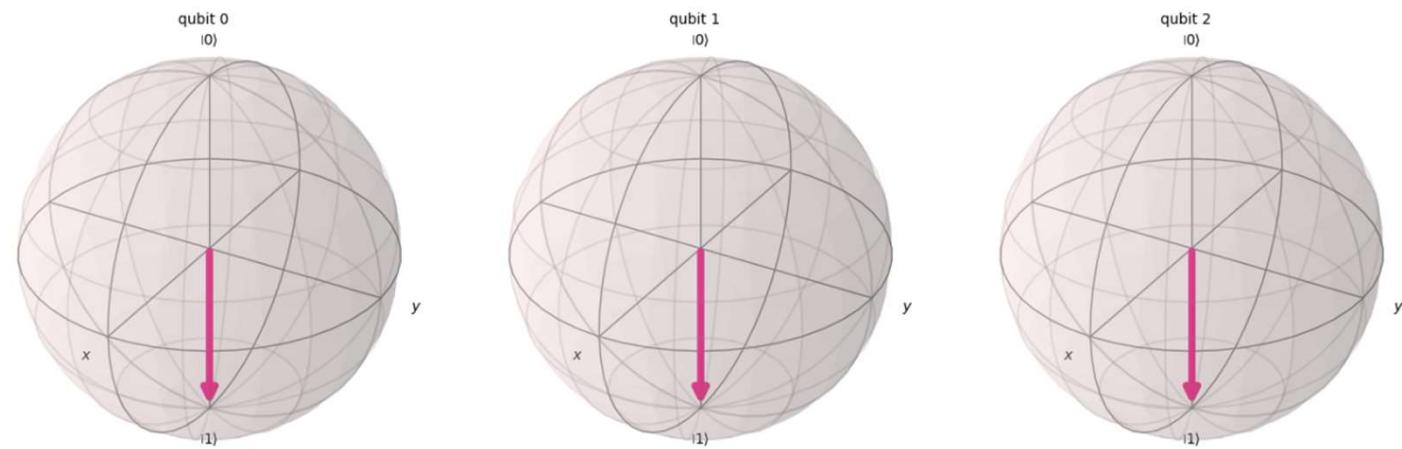
[19]:



## Toffoli Gate

```
[20]: bloch_sphere
```

```
[20]:
```



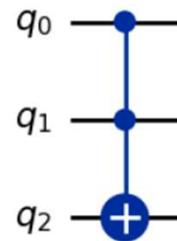
## Toffoli Gate Decompose

```
#####
# CURSO QISKIT 2025
# Multi Qubit TOFFOLI DECOMPOSED
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#-----
def Diagramas(quantum_circuit):
    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    qc_decomposed = quantum_circuit.decompose()
    Decomp = qc_decomposed.draw(output='mpl')
    return circuit_diagram, Decomp
#-----
qc = QuantumCircuit(3)
qc.ccx(0,1,2)
Circuito, Circuito_decomp = Diagramas(qc)
#####
```

# Toffoli Gate Decompose

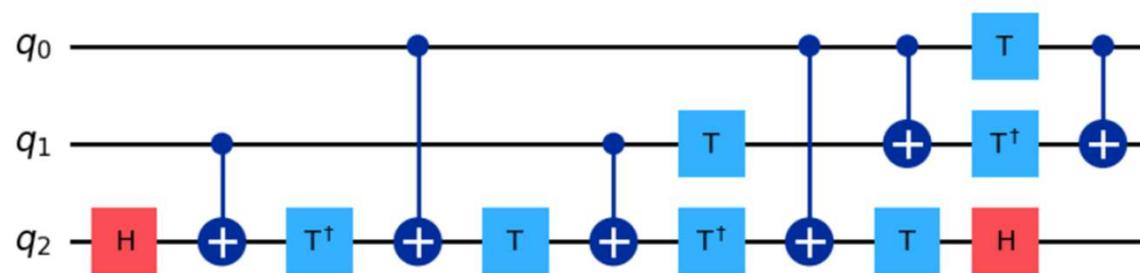
[22]: Circuito

[22]:

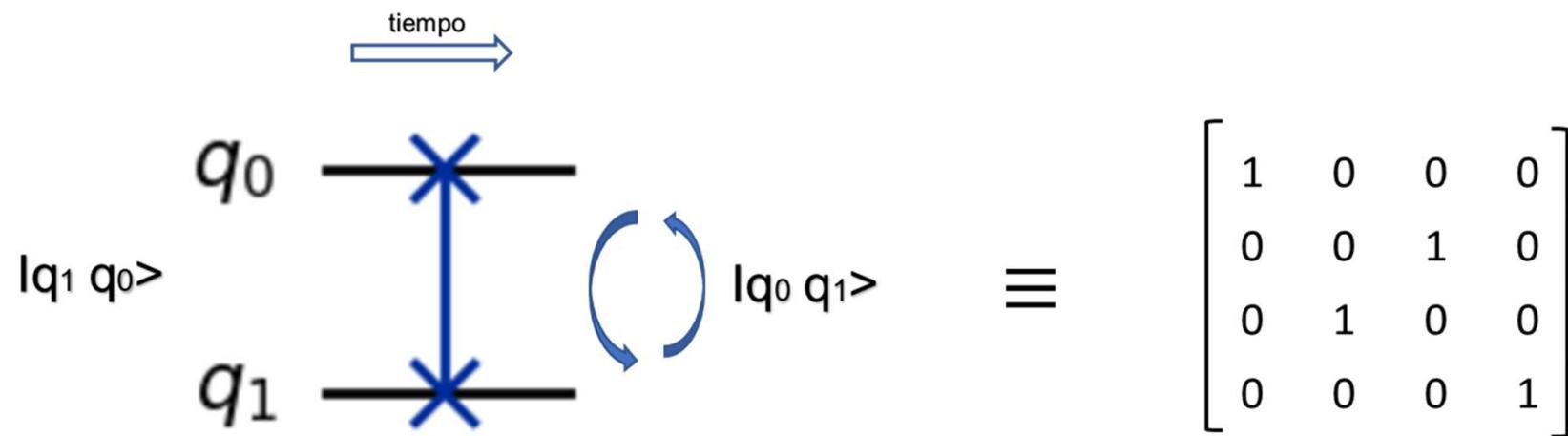


[23]: Circuito\_decomp

[23]:



## SWAP Gate



## SWAP Gate

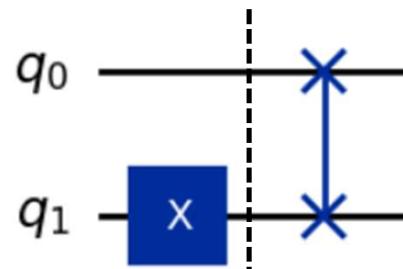
```
#####
# CURSO QISKIT 2025
# Multi Qubit SWAP GATE
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import *
from math import pi
#-----
# Execute circuit on the the State vector sampler
def run_sv_circuit(qc):
    # Run the circuit and return the state vector object result
    stateVectorResult = Statevector(qc)
    #qsphere = stateVectorResult.draw('qsphere')
    #bloch_sphere = stateVectorResult.draw('bloch')
    #circuit_image = qc.draw(output='mpl')
    return stateVectorResult
#-----
# Will run the circuit on the state vector (sv) simulator
# Returns state vector results, circuit diagram, Sphere & Bloch sphere
def execute_circuit_sv(quantum_circuit):
    stateVectorResults = run_sv_circuit(quantum_circuit)
    #Draw the circuit diagram
    circuit_diagram = quantum_circuit.draw(output="mpl")
    #Draw the Qsphere
    q_sphere = stateVectorResults.draw('qsphere')
    #Draw the Bloch sphere
    bloch_sphere = stateVectorResults.draw('bloch')
    #Return the results, circuit diagram, and QSphere
    return stateVectorResults, circuit_diagram, q_sphere, bloch_sphere
#-----
qc = QuantumCircuit(2)
qc.x(1)
qc.swap(0,1)
result, img, qsphere, bloch_sphere = execute_circuit_sv(qc)
#####
```

## SWAP Gate

[3]: img

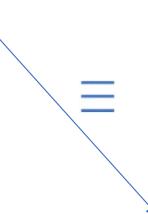
[3]:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$$|01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$|10\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$



[2]: result

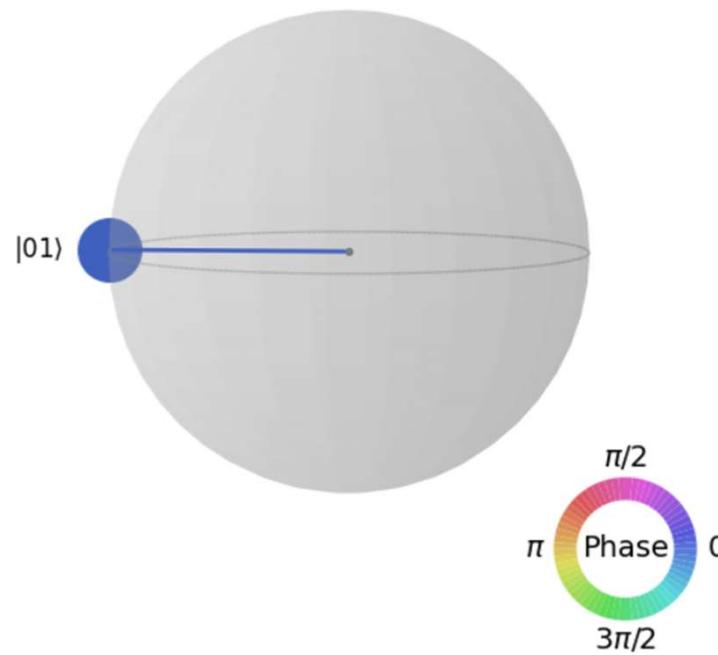
```
Statevector([0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j],  
           dims=(2, 2))
```

$$|01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

SWAP Gate

[4]: qsphere

[4]:

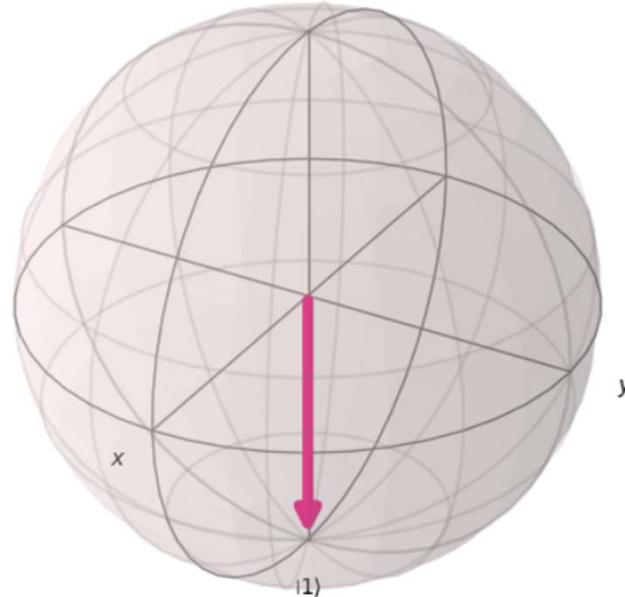


## SWAP Gate

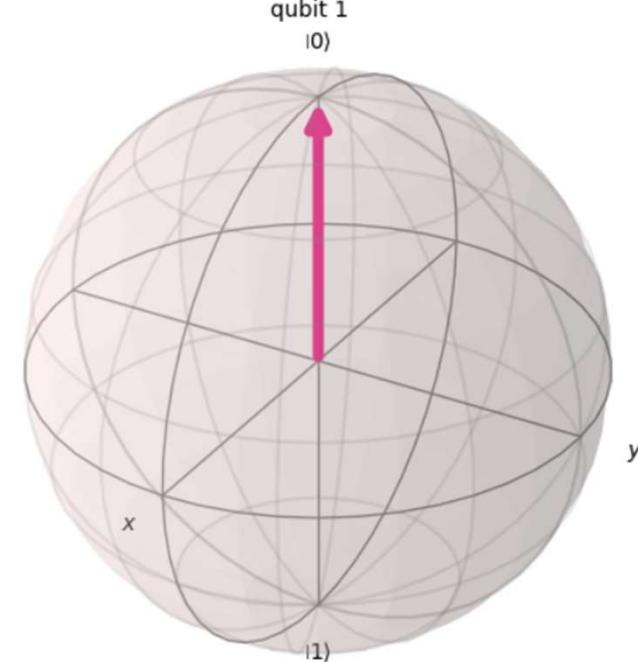
```
[5]: bloch_sphere
```

```
[5]:
```

qubit 0  
 $|0\rangle$



qubit 1  
 $|0\rangle$



## Constructores

```
#####
# CURSO QISKIT 2025
# Constructors

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

qc = QuantumCircuit(2,2)
qr = QuantumRegister(2,'Mi_QR')
cr = ClassicalRegister(2,'Mi_CR')
qc = QuantumCircuit(qr,cr)
qc.draw(output='mpl')
#####
```

## Constructores

[1]:

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

qc = QuantumCircuit(2,2)
qr = QuantumRegister(2,'Mi_QR')
cr = ClassicalRegister(2,'Mi_CR')
qc = QuantumCircuit(qr,cr)
qc.draw(output='mpl')
```

[1]:

*Mi\_QR<sub>0</sub>* —

*Mi\_QR<sub>1</sub>* —

*Mi\_CR*  $\frac{2}{\cancel{\cancel{ }}} \cancel{\cancel{ }}$

## Constructores Alternativa Compacta

```
#####
# CURSO QISKIT 2025
# Constructors

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

qc = QuantumCircuit(2,2)
qc = QuantumCircuit(QuantumRegister(2,'Mi_QR'),ClassicalRegister(2,'Mi_CR'))
qc.draw(output='mpl')
#####
```

## Constructores Alternativa Compacta

```
[2]: from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister  
  
qc = QuantumCircuit(2,2)  
qc = QuantumCircuit(QuantumRegister(2,'Mi_QR'),ClassicalRegister(2,'Mi_CR'))  
qc.draw(output='mpl')
```

```
[2]:
```

*Mi\_QR<sub>0</sub>* —

*Mi\_QR<sub>1</sub>* —

*Mi\_CR* 

## Circuitos Combinados

```
#####
# CURSO QISKIT 2025
# COMBINAR CIRCUITOS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

#-----
def Dibujar (qc_1, qc_2, qc_combined):
    #Draw circuit diagram #1
    circuito_1 = qc_1.draw(output='mpl')
    #Draw circuit diagram #2
    circuito_2 = qc_2.draw(output='mpl')
    #Draw circuit combinado
    combinado = qc_combined.draw(output='mpl')
    #Return the results
    return circuito_1, circuito_2, combinado
#-----

qr1 = QuantumRegister(2,'Mi_qr_1')
cr1 = ClassicalRegister(2,'Mi_cr_1')
qc1 = QuantumCircuit(qr1,cr1)

qr2 = QuantumRegister(2,'Mi_qr_2')
cr2 = ClassicalRegister(2,'Mi_cr_2')
qc2 = QuantumCircuit(qr2,cr2)
qc2.draw(output='mpl')

qc_combined = QuantumCircuit()
qc_combined.add_register(qr1,qr2,cr1,cr2)

circ_1, circ_2, circ_combined= Dibujar(qc1, qc2, qc_combined)
#####
```

## Circuitos Combinados

[7]: circ\_1

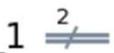
[8]: circ\_2

[9]: circ\_combined

[7]:

[8]:

[9]:

 $Mi\_qr\_1_0$  — $Mi\_qr\_2_0$  — $Mi\_qr\_1_0$  — $Mi\_qr\_1_1$  — $Mi\_qr\_2_1$  — $Mi\_qr\_1_1$  — $Mi\_cr\_1$   $Mi\_cr\_2$   $Mi\_qr\_2_0$  — $Mi\_qr\_2_1$  — $Mi\_cr\_1$   $Mi\_cr\_2$  

## RANDOM CIRCUITS GENERATORS

Width = 3 nro Qubits  
Depth = 2 nro Std Gates

```
#####
# CURSO QISKIT 2025
# RANDOM CIRCUITS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.random import random_circuit

qc = random_circuit(3,2,measure=True)
qc.draw(output='mpl')

#####
```

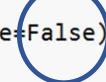
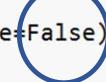
← Con Medidores →

```
#####
# CURSO QISKIT 2025
# RANDOM CIRCUITS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.random import random_circuit

qc = random_circuit(3,2,measure=False)
qc.draw(output='mpl')

#####
```



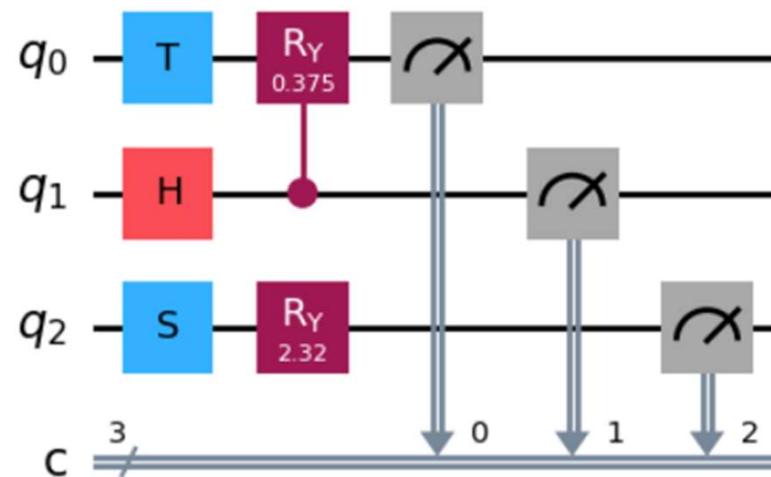
→ Sin Medidores

## RANDOM CIRCUITS GENERATORS

```
[12]: from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister  
from qiskit.circuit.random import random_circuit  
  
qc = random_circuit(3,2,measure=True)  
qc.draw(output='mpl')
```

[12]:

Ejecución #1  
Con Medición  
Width = 3  
Depth = 2

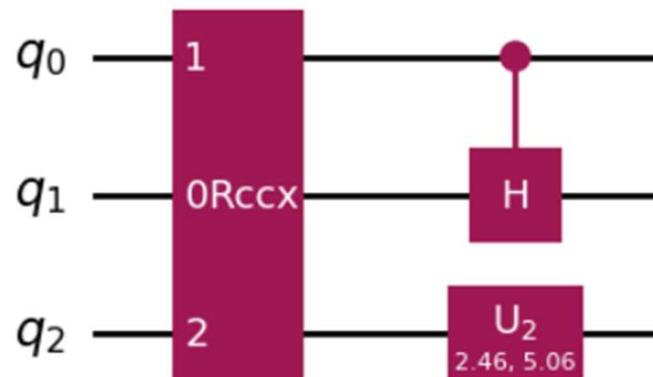


## RANDOM CIRCUITS GENERATORS

```
[14]: from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister  
from qiskit.circuit.random import random_circuit  
  
qc = random_circuit(3,2,measure=False)  
qc.draw(output='mpl')
```

[14]:

Ejecución #2  
Sin Medición  
Width = 3  
Depth = 2



## Obtención Propiedades del Circuito

WIDTH:

Cantidad de Qubits + Classical Bits

DEPTH:

Número de Standard Gates por Quibit (por línea)

CIRCUIT SIZE:

Número total de Gates

OPERATOR COUNT:

Listado de cada gate y su cantidad.

## Obtención Propiedades del Circuito

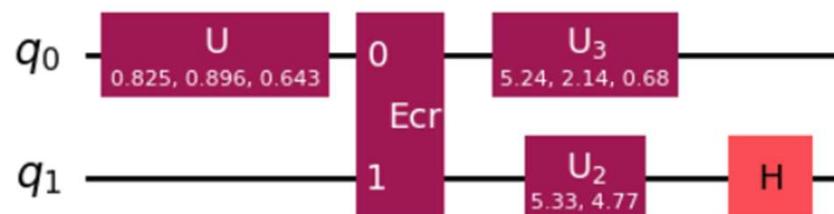
```
#####
# CURSO QISKIT 2025
# WIDTH - DEPTH - CIRCUIT SIZE - NUMBER OPERATORS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.random import random_circuit

qc1 = random_circuit(2,2)
qc2 = random_circuit(2,2)
#concatenar
qc=qc1.compose(qc2,[0,1])
qc.draw(output='mpl')

#####
```

[3]:



Genero  
Circuito  
RANDOM

## Obtención Propiedades del Circuito

```
#####
# CURSO QISKIT 2025
# WIDTH - DEPTH - CIRCUIT SIZE - NUMBER OPERATORS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.random import random_circuit

#-----
#Defino Funcion print propiedades
def print_circuit_props(qc):
    width = qc.width()
    depth = qc.depth()
    num_operators = qc.count_ops()
    circuit_size = qc.size()
    print('WIDTH = ', width)
    print('DEPTH = ', depth)
    print('Circuit Size =', circuit_size)
    print('Numero Operadores = ', num_operators)
#-----
qc1 = random_circuit(2,2)
qc2 = random_circuit(2,2)
#concatenar
qc=qc1.compose(qc2,[0,1])
qc.draw(output='mpl')
#####

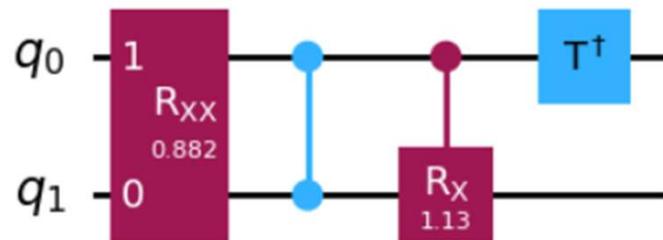
#####
# Luego ejecutar la impresion de propiedades

print_circuit_props(qc)

#####
```

## Obtención Propiedades del Circuito

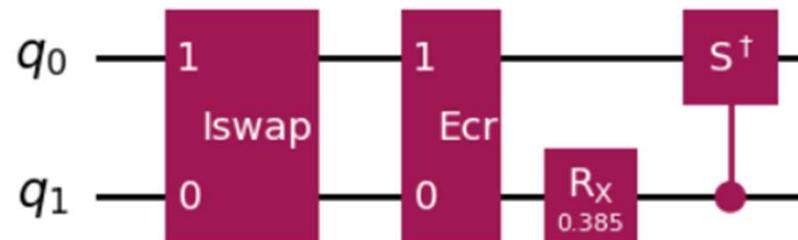
[11]:

Ejecución # 1  
Sin Medicion[12]: `print_circuit_props(qc)`

```
WIDTH = 2
DEPTH = 4
Circuit Size = 4
Número Operadores = OrderedDict({'rxx': 1, 'cz': 1, 'crx': 1, 'tdg': 1})
```

## Obtención Propiedades del Circuito

[13]:

Ejecución # 2  
Sin medicion[14]: `print_circuit_props(qc)`

```
WIDTH = 2
DEPTH = 4
Circuit Size = 4
Número Operadores = OrderedDict({'iswap': 1, 'ecr': 1, 'rx': 1, 'csdg': 1})
```

## Obtención Propiedades del Circuito

```
#####
# CURSO QISKIT 2025
# WIDTH - DEPTH - CIRCUIT SIZE - NUMBER OPERATORS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.random import random_circuit

#-----
#Defino Funcion print propiedades
def print_circuit_props(qc):
    width = qc.width()
    depth = qc.depth()
    num_operators = qc.count_ops()
    circuit_size = qc.size()
    print('WIDTH = ', width)
    print('DEPTH = ', depth)
    print('Circuit Size =', circuit_size)
    print('Numero Operadores = ', num_operators)
#-----
qc1 = random_circuit(3,4)
qc2 = random_circuit(3,4,measure=True)
#concatenar
qc=qc1.compose(qc2,[0,1,2])
qc.draw(output='mpl')
#####

#####
# Luego ejecutar la impresion de propiedades

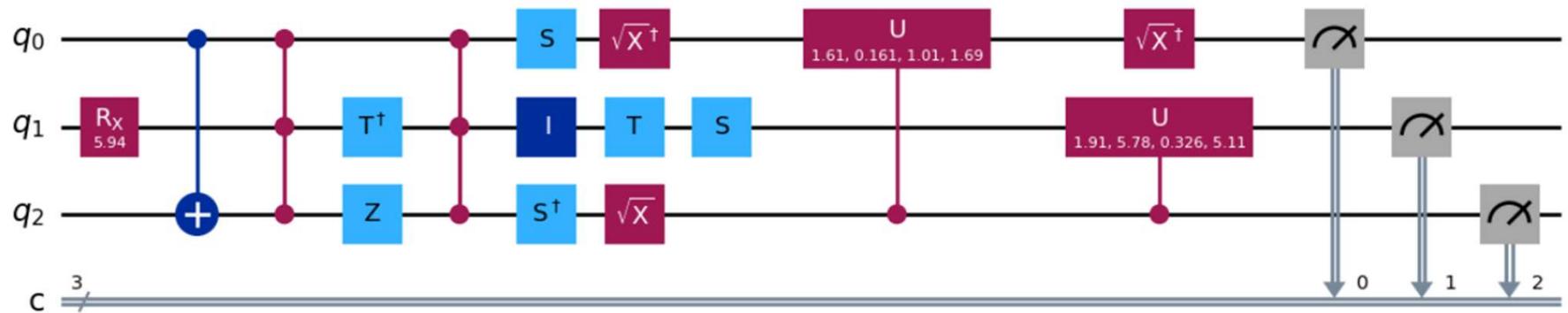
print_circuit_props(qc)

#####
```

## Obtención Propiedades del Circuito

Ejecución # 1  
Con medición

[20]:

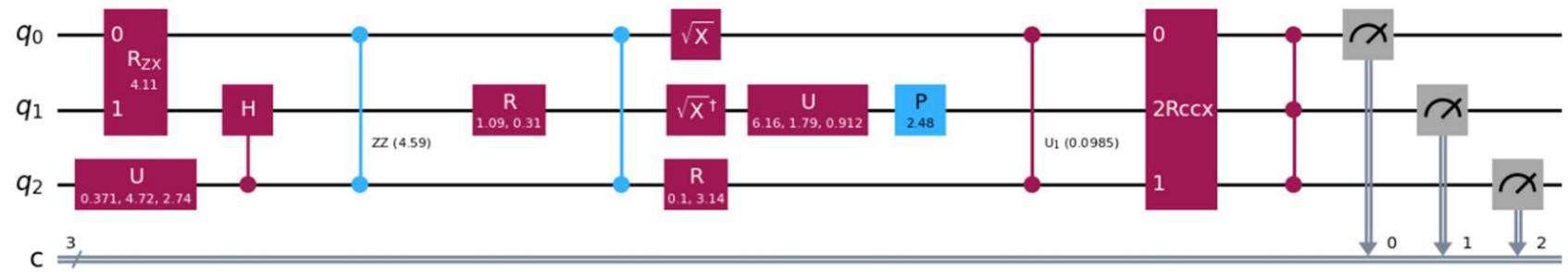
[21]: `print_circuit_props(qc)`

```
WIDTH = 6
DEPTH = 9
Circuit Size = 19
Número Operadores = OrderedDict({'measure': 3, 'ccz': 2, 's': 2, 'sxdg': 2, 'cu': 2, 'rx': 1, 'cx': 1, 'tdg': 1, 'z': 1, 'id': 1, 'sdg': 1, 'sx': 1, 't': 1})
```

## Obtención Propiedades del Circuito

Ejecución # 2  
Con medición

[22]:

[23]: `print_circuit_props(qc)`

```
WIDTH = 6
DEPTH = 9
Circuit Size = 17
Número Operadores = OrderedDict({'measure': 3, 'u': 2, 'r': 2, 'rzx': 1, 'ch': 1, 'rzz': 1, 'cz': 1, 'sx': 1, 'sxg': 1, 'p': 1, 'cu1': 1, 'rccx': 1, 'ccz': 1})
```

## Obtención Propiedades TOFFOLI

```
#####
# CURSO QISKIT 2025
# WIDTH - DEPTH - CIRCUIT SIZE - NUMBER OPERATORS

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.random import random_circuit

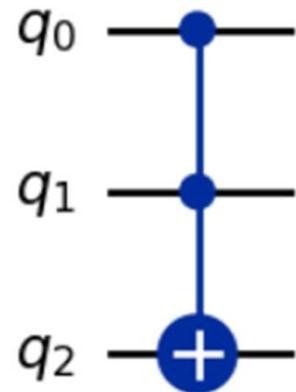
#-----
#Defino Funcion print propiedades
def print_circuit_props(qc):
    width = qc.width()
    depth = qc.depth()
    num_operators = qc.count_ops()
    circuit_size = qc.size()
    print('WIDTH = ', width)
    print('DEPTH = ', depth)
    print('Circuit Size =', circuit_size)
    print('Numero Operadores = ', num_operators)
#-----
qc = QuantumCircuit(3)
qc.ccx(0,1,2)
qc.draw(output='mpl')
#####

#####
# Luego ejecutar la impresion de propiedades

print_circuit_props(qc)
#####
```

## Obtención Propiedades TOFFOLI

[24]:

[25]: `print_circuit_props(qc)`

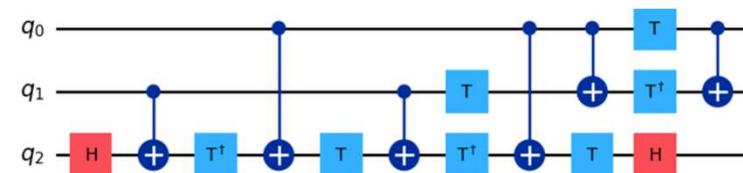
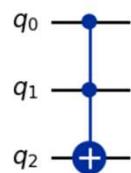
```
WIDTH = 3
DEPTH = 1
Circuit Size = 1
Número Operadores = OrderedDict({'ccx': 1})
```

## Obtención Propiedades TOFFOLI Decompose

Toffoli se compone de varias Basis Gates

```
[27]: print_circuit_props(qc.decompose())
```

```
WIDTH = 3
DEPTH = 11
Circuit Size = 15
Número Operadores = OrderedDict({'cx': 6, 't': 4, 'tdg': 3, 'h': 2})
```



## Lista Instrucciones - COMPOSITE

```
#####
# CURSO QISKIT 2025
# COMPOSITE LISTA INSTRUCCIONES
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
#-----
#Creacion Lista Instrucciones COMPOSITE
qr = QuantumRegister(2, name= 'qr_c')
comp_qc = QuantumCircuit(qr, name='Mi Composite')
comp_qc.h(0)
comp_qc.cx(0,1)
#creo lista de instrucciones del quantumCircuit
composite_inst = comp_qc.to_instruction()
comp_qc.draw(output='mpl')
#####

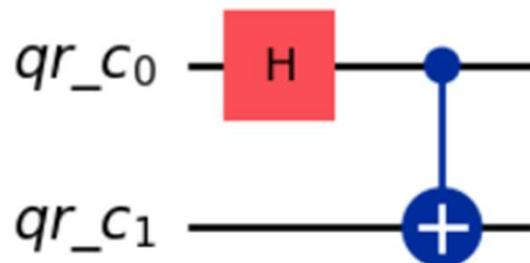
#####
#Creacion nuevo circuito
qr2 = QuantumRegister(3,'qr')
qc = QuantumCircuit(qr2)
qc.h(0)
qc.cx(0,1)
qc.cx(0,2)
qc.draw(output='mpl')
#####

#####
#Append el compuesto tomado del listado de instrucciones
qc.append(composite_inst, [qr2[0], qr2[1]])
qc.draw(output='mpl')
#####
```

## Lista Instrucciones - COMPOSITE

```
[2]: from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister  
#-----  
#Creacion Lista Instrucciones COMPOSITE  
qr = QuantumRegister(2, name= 'qr_c')  
comp_qc = QuantumCircuit(qr, name='Mi Composite')  
comp_qc.h(0)  
comp_qc.cx(0,1)  
#creo lista de instrucciones del quantumCircuit  
composite_inst = comp_qc.to_instruction()  
comp_qc.draw(output='mpl')
```

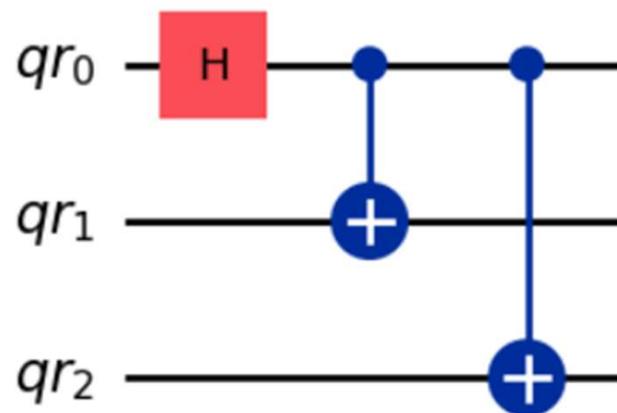
```
[2]:
```



## Lista Instrucciones - COMPOSITE

```
[3]: #Creacion nuevo circuito
qr2 = QuantumRegister(3, 'qr')
qc = QuantumCircuit(qr2)
qc.h(0)
qc.cx(0,1)
qc.cx(0,2)
qc.draw(output='mpl')
```

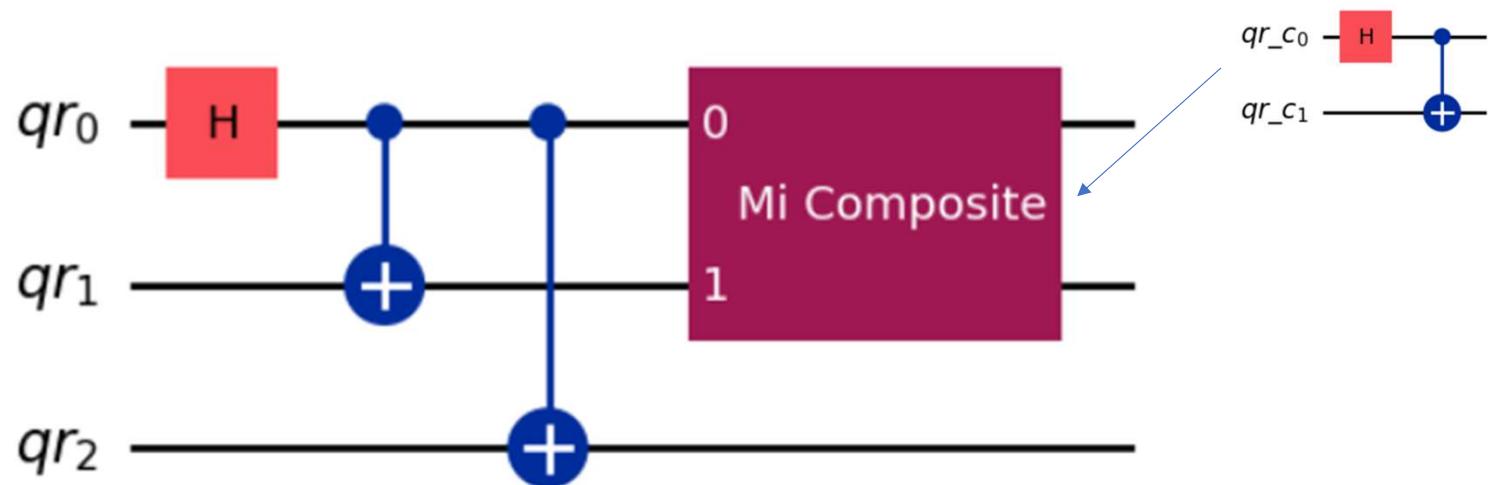
```
[3]:
```



## Lista Instrucciones - COMPOSITE

```
[4]: #Append el compuesto tomado del listado de instrucciones  
qc.append(composite_inst, [qr2[0], qr2[1]])  
qc.draw(output='mpl')
```

```
[4]:
```

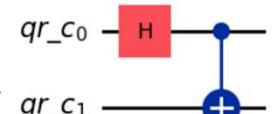
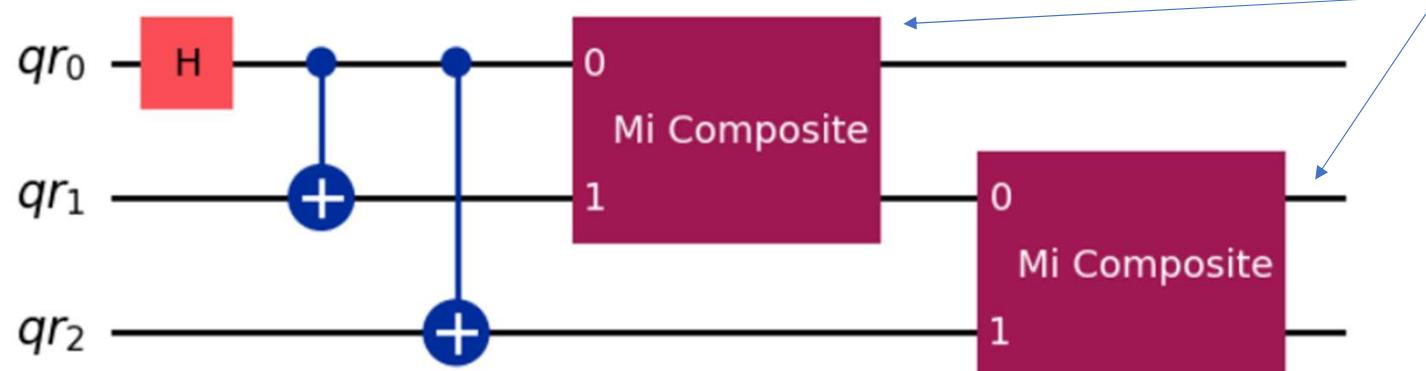


## Lista Instrucciones - COMPOSITE

Ejecuto nuevamente el append del composite para los qubits 1 y 2

```
[5]: #Append el compuesto tomado del listado de instrucciones  
qc.append(composite_inst, [qr2[1], qr2[2]])  
qc.draw(output='mpl')
```

[5]:



## INTRODUCCIÓN

**Aparenta** desperdicio  
recursos y tiempo

- El problema que resuelve es muy simple
- Problema inventado. Jamás lo encontramos en el mundo real
- La técnica utilizada por este algoritmo no tiene semejanza a los procedimientos paso a paso aplicados en computación clásica

**Como prueba de concepto, el algoritmo de Deutsch es un gran salto hacia adelante.**

## DESCRIPCIÓN

Resuelve un problema relacionado a funciones:

BALANCEADAS  
CONSTANTES

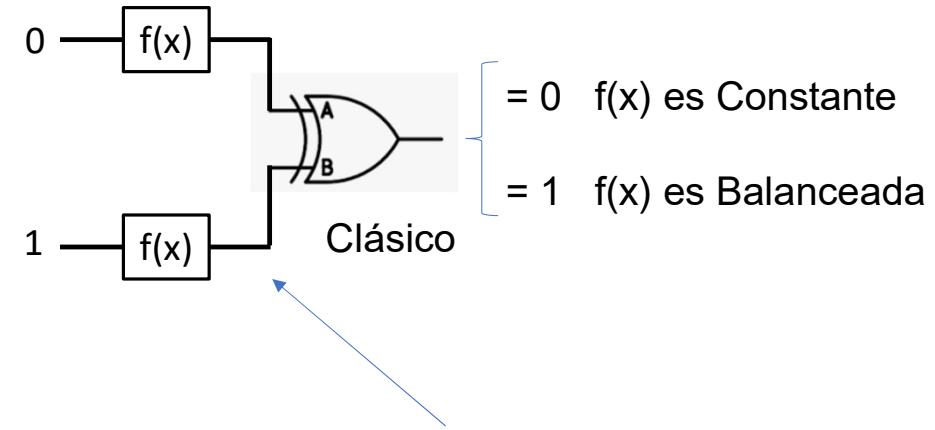
## DESCRIPCIÓN Funciones Constantes y Balanceadas

El problema que resuelve ilustra la forma como mediante un circuito cuántico se pueden resolver varios (en este caso dos) pasos, o evaluaciones, en uno aplicando superposición.

Función: { Constante  
o  
Balanceada

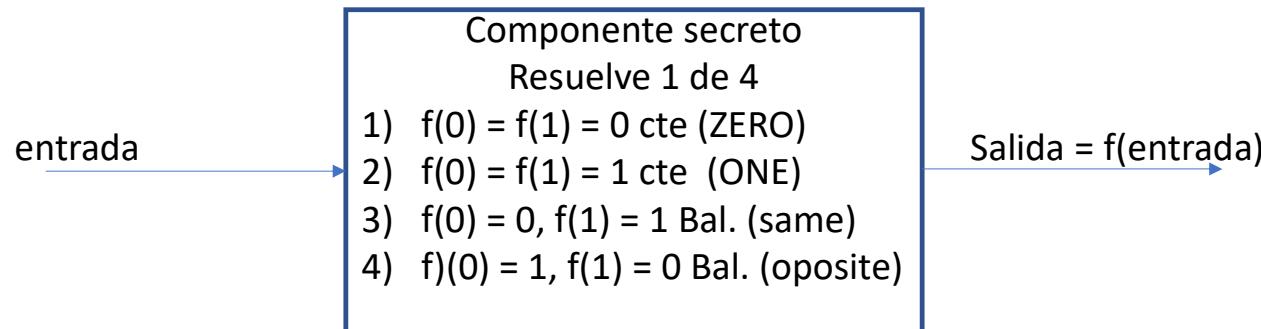
Constante: {  $f(0) = f(1) = 0$  (ZERO)  
o  
 $f(0) = f(1) = 1$  (ONE)

Balanceada: {  $f(0) = 0$  y  $f(1) = 1$  (Same\_As)  
o  
 $f(0) = 1$  y  $f(0) = 0$  (Opposite\_Of)



Clásico: requiere evaluar la función **2 veces**.

## Resolución forma clásica



Input es 0 y almaceno la salida en variable a  
Input es 1 y almaceno la salida en variable b  
Si  $a = b$

Función Constante

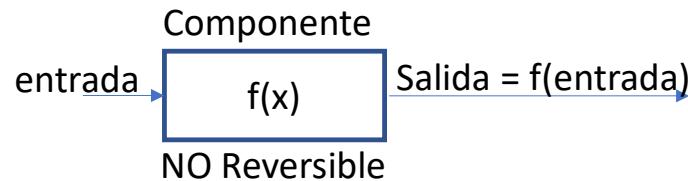
Si no

Función Balanceada

Evaluación en 2 pasos

Componente Secreto  $F(x)$  debe ser reversible

Debido a la propiedad de REVERSIBILIDAD de las compuertas cuánticas, el componente secreto que resuelve la función  $y$  que deseamos conocer si es una función Balanceada o Constante, debe ser **REVERSIBLE**



| Entrada | Salida |
|---------|--------|
| 0       | 0      |
| 1       | 0      |

Ejemplo  
Si  $f(x)$  es CTE (ZERO)



Seteando  $y = 0$  (ancilla), la salida inferior es  
 $|f(x) \oplus y\rangle = |f(x) \oplus 0\rangle = |f(x)\rangle$

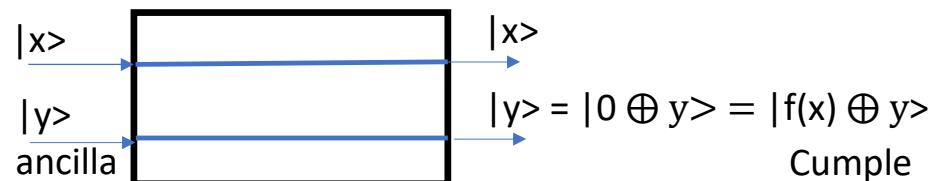
## DEUTSCH'S ALGORITHM

# Quantum Computing QISKit

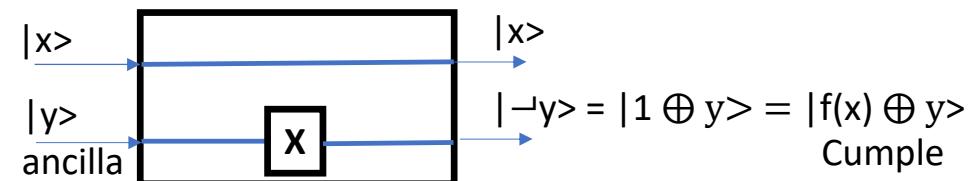
Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

Componente Secreto  $F(x)$  debe ser reversible

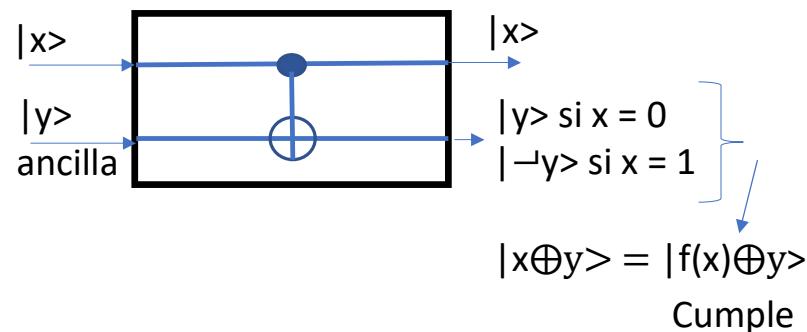
Función ZERO  $f(x) = 0$  siempre.



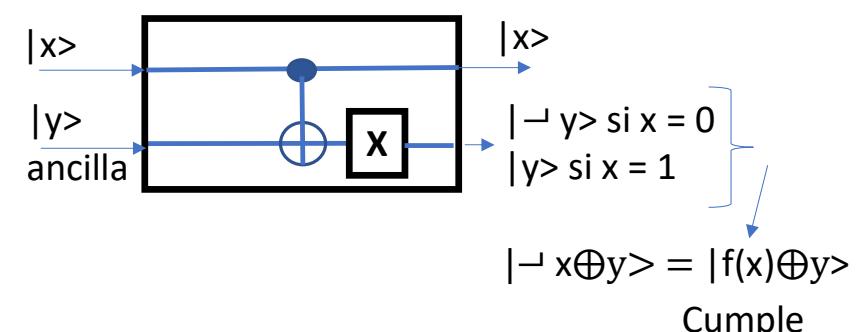
Función ONE  $f(x) = 1$  siempre.



Función Same as  $f(x) = x$ .



Función Opposite of as  $f(x) = \neg x$ .

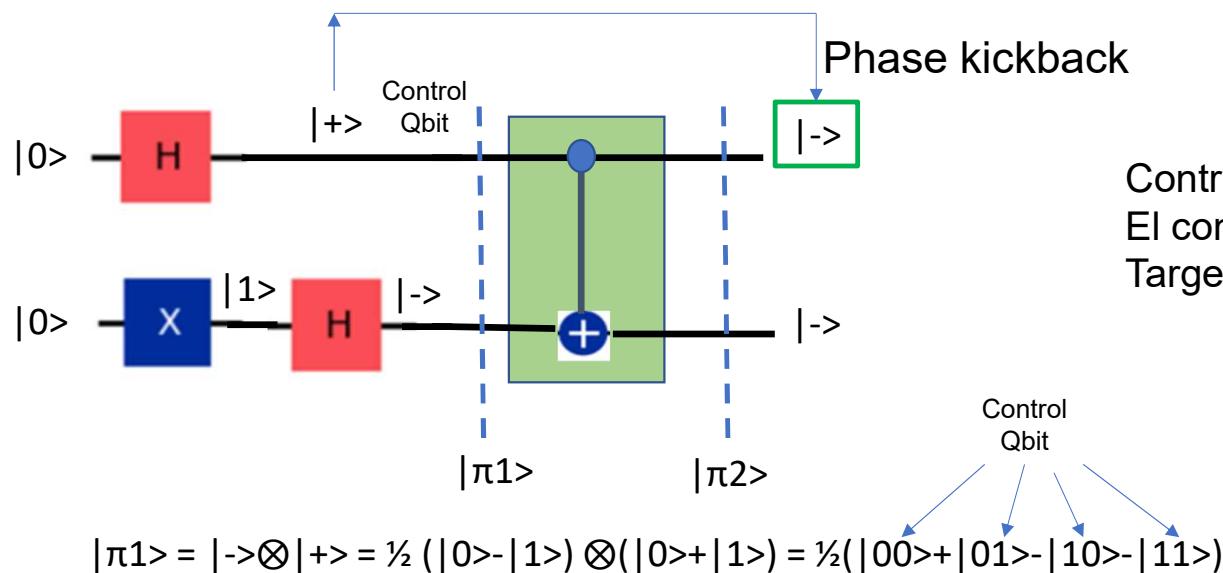


168



CamScanner

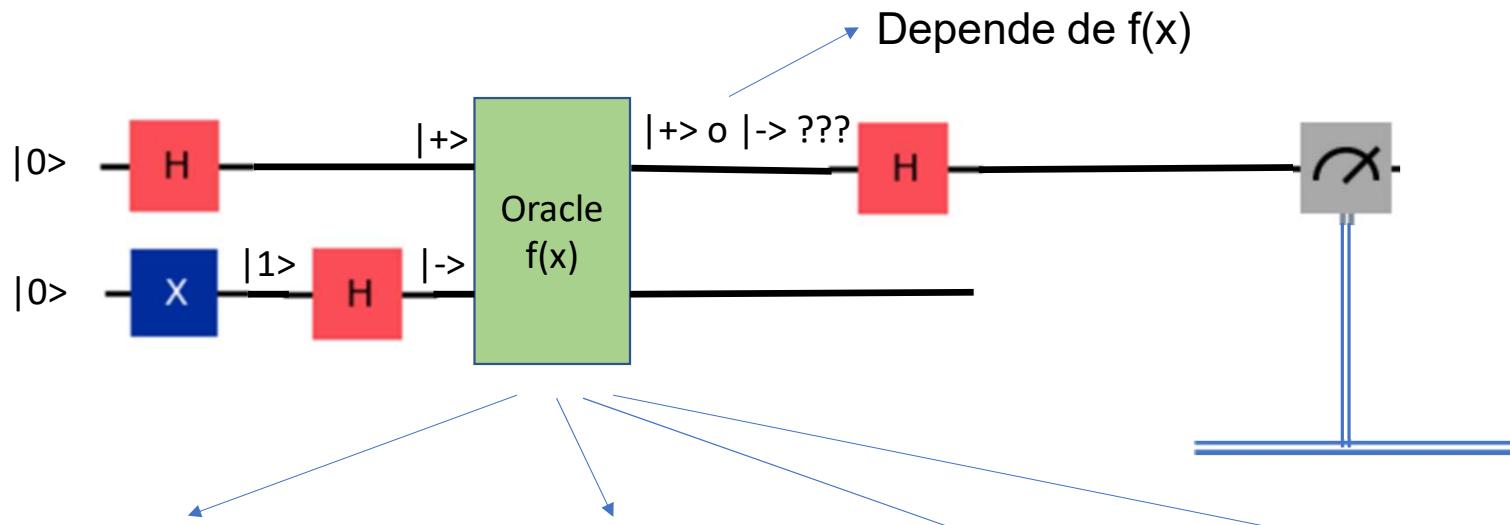
## Phase Kickback



$$|\pi_2\rangle = |-\rangle \otimes |+\rangle = \frac{1}{2} (|00\rangle + |11\rangle - |10\rangle - |01\rangle) = \frac{1}{2}(|0\rangle - |1\rangle) \otimes (|0\rangle - |1\rangle) = |-\rangle \otimes |-\rangle$$

$$|\text{Salida}\rangle = |-\rangle \otimes |-\rangle = |-\rangle |-\rangle$$

## ORÁCULO

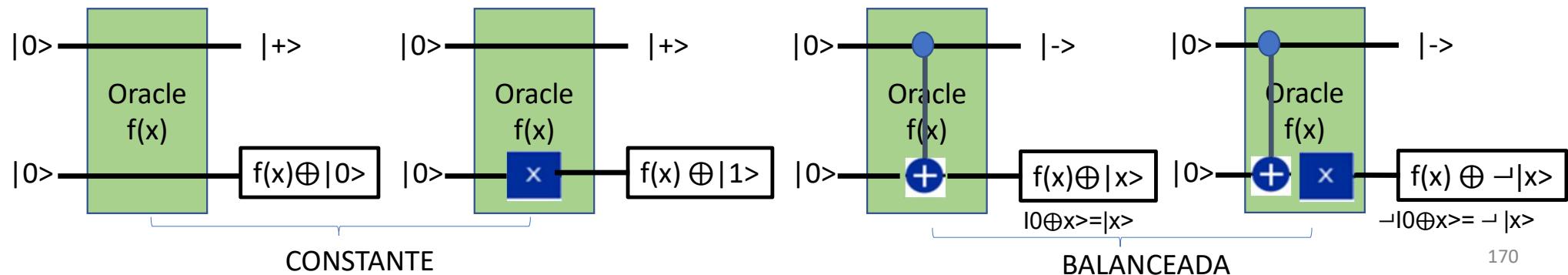


ZERO

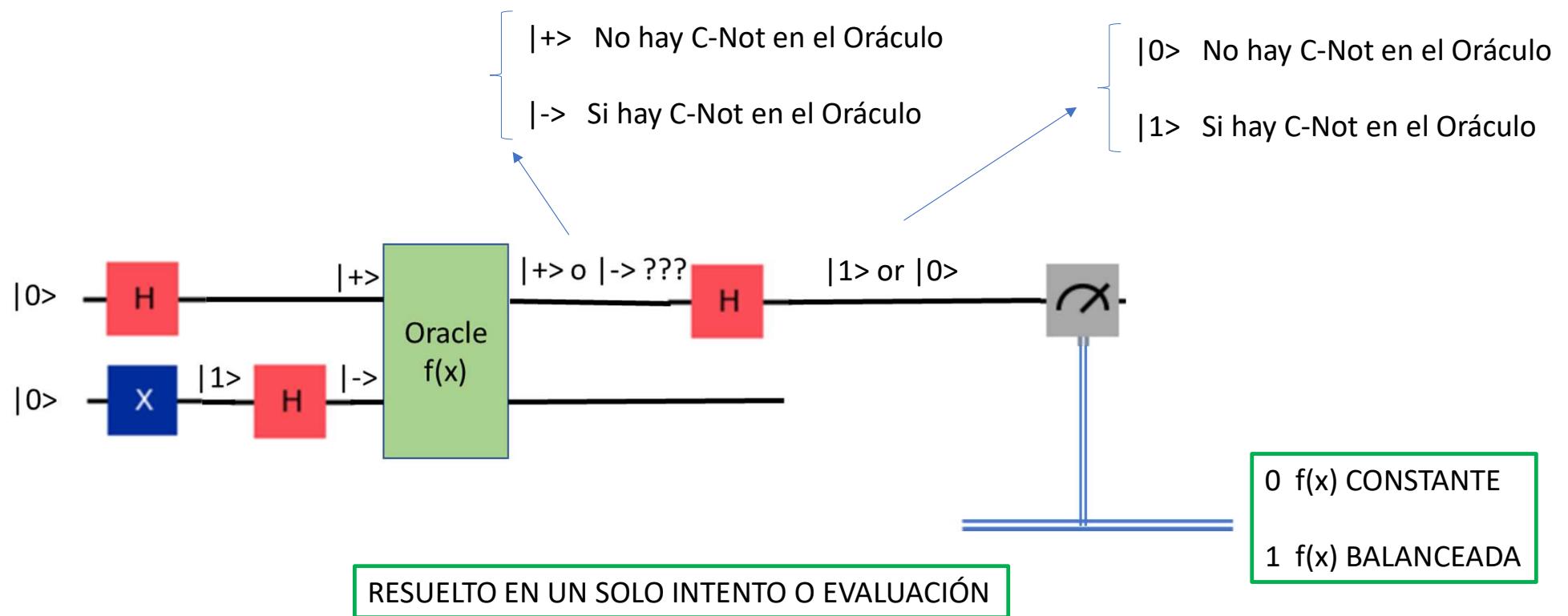
ONE

Same\_of

Opposite\_of



## ORÁCULO



## Programa QuisKit

```
#####
# CURSO QISKIT 2025
# DEUTSCH Algorithm

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from enum import Enum
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_bloch_multivector

qc = QuantumCircuit(2,2)
qr = QuantumRegister(2,'Mi_QR')
cr = ClassicalRegister(2,'Mi_CR')
qc = QuantumCircuit(qr,cr)
qc.draw(output='mpl')
#-----
class SimpleBinary(Enum):
    ZERO      = 0
    ONE       = 1
    SAME_AS   = 2
    OPPOSITE_OF = 3
#-----
def get_oracle(circ, function):
    # if function == SimpleBinary.ZERO:
    #     # Do nothing
    if function == SimpleBinary.ONE:
        circ.x(1)
    elif function == SimpleBinary.SAME_AS:
        circ.cx(0, 1)
    elif function == SimpleBinary.OPPOSITE_OF:
        circ.cx(0, 1)
        circ.x(1)
    return circ
#-----
```

## Programa QuisKit

```
#-----
def get_function():
    print('Which function? (0/1/2/3)')
    print(' 0: ZERO')
    print(' 1: ONE')
    print(' 2: SAME_AS')
    print(' 3: OPPOSITE_OF')
    value = input('> ')
    return SimpleBinary(int(value))
#-----
circ = QuantumCircuit(2, 1)
function = get_function()
circ.x(1)
circ.h(0)
circ.h(1)
circ.barrier()
circ = get_oracle(circ, function)
circ.barrier()
circ.h(0)
circ.measure(0, 0)
display(circ.draw('mpl'))
```

## Programa QuisKit

```
from qiskit import transpile
from qiskit_aer import Aer
from qiskit_aer import AerSimulator

shots = 1
simulator = AerSimulator()
compiled_circuit = transpile(circ, simulator)
sim_result = simulator.run(compiled_circuit, shots=1).result()
counts = sim_result.get_counts()
print(counts)
print(function)
print(counts)
number_of_0s = counts.get('0')
number_of_1s = counts.get('1')

if number_of_0s is not None and number_of_0s == shots:
    print('Constant')
elif number_of_1s is not None and number_of_1s == shots:
    print('Balanced')
else:
    print("Results aren't conclusive")
#####
```

## Ejecución Programa QuisKit

```
Which function? (0/1/2/3)
```

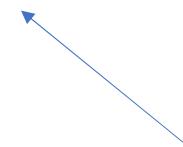
```
0: ZERO
```

```
1: ONE
```

```
2: SAME_AS
```

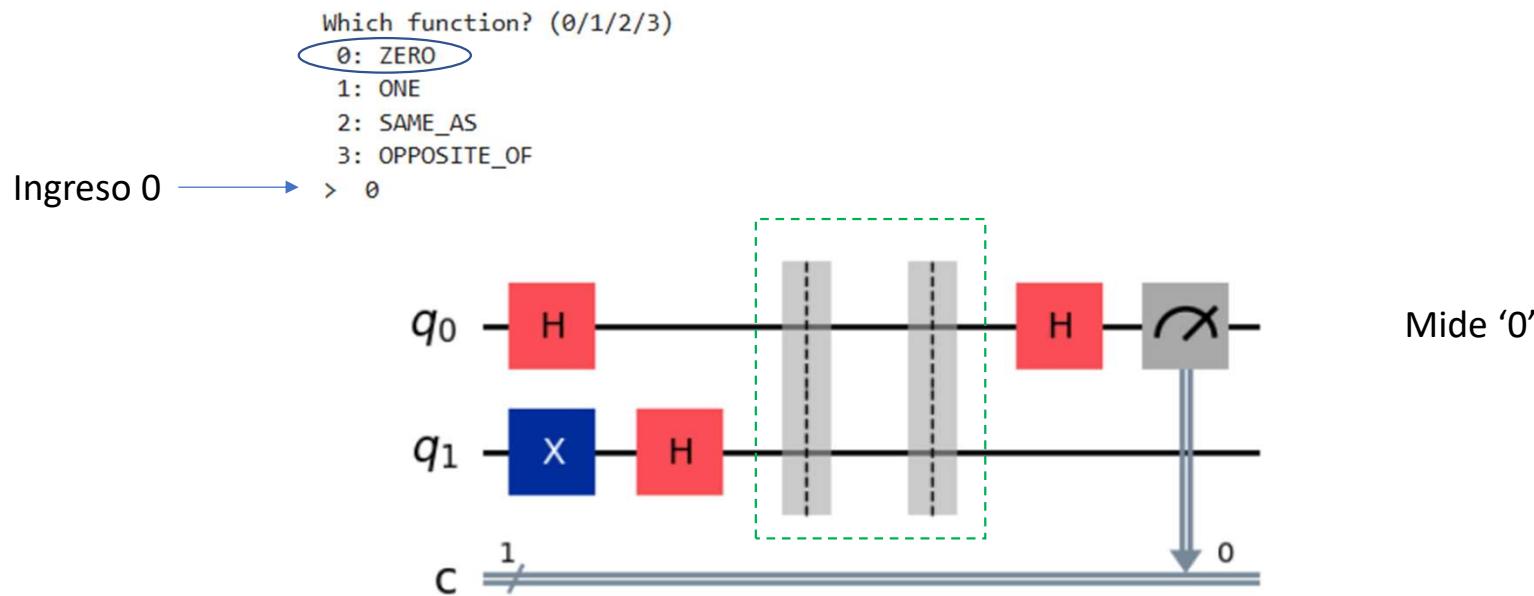
```
3: OPPOSITE_OF
```

```
> ↑↓ for history. Search history with c-↑/c-↓
```



Ingresar 0, 1, 2 o 3

## Ejecución Programa QuisKit



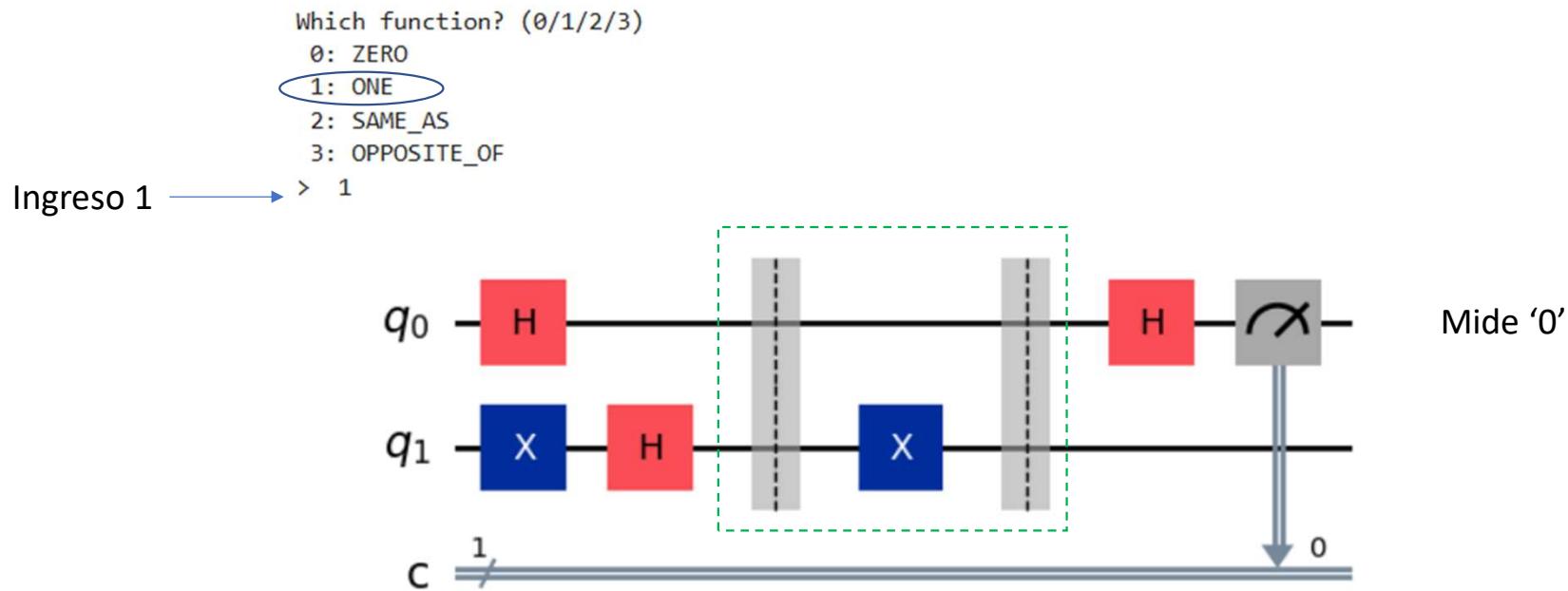
```
{'0': 1}  
SimpleBinary.ZERO  
{'0': 1}  
Constant
```

## DEUTSCH'S ALGORITHM

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Ejecución Programa QuisKit



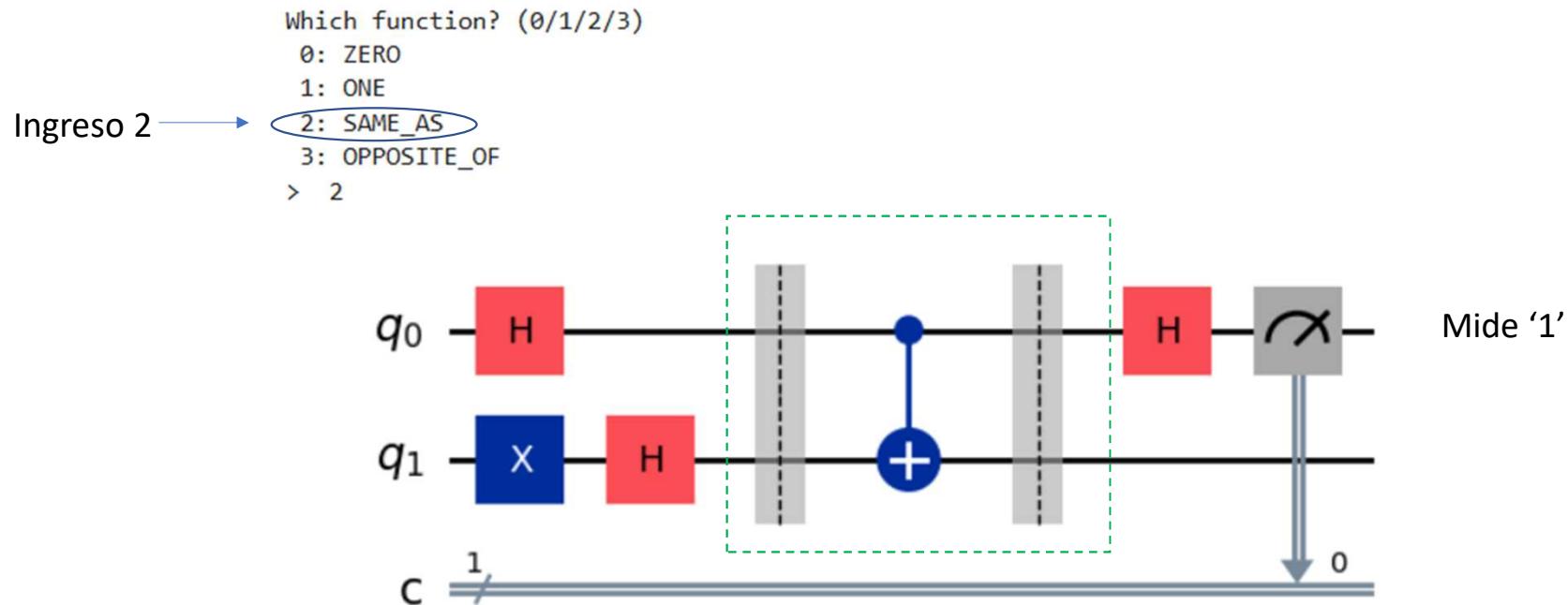
```
{'0': 1}  
SimpleBinary.ONE  
{'0': 1}  
Constant
```

## DEUTSCH'S ALGORITHM

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Ejecución Programa QuisKit



```
{'1': 1}  
SimpleBinary.SAME_AS  
{'1': 1}  
Balanced
```

178

## DEUTSCH'S ALGORITHM

# Quantum Computing QISKit

Ing. Marcos FRANKIENSZTAJN 16-DIC-2025

## Ejecución Programa QuisKit

Ingreso 3

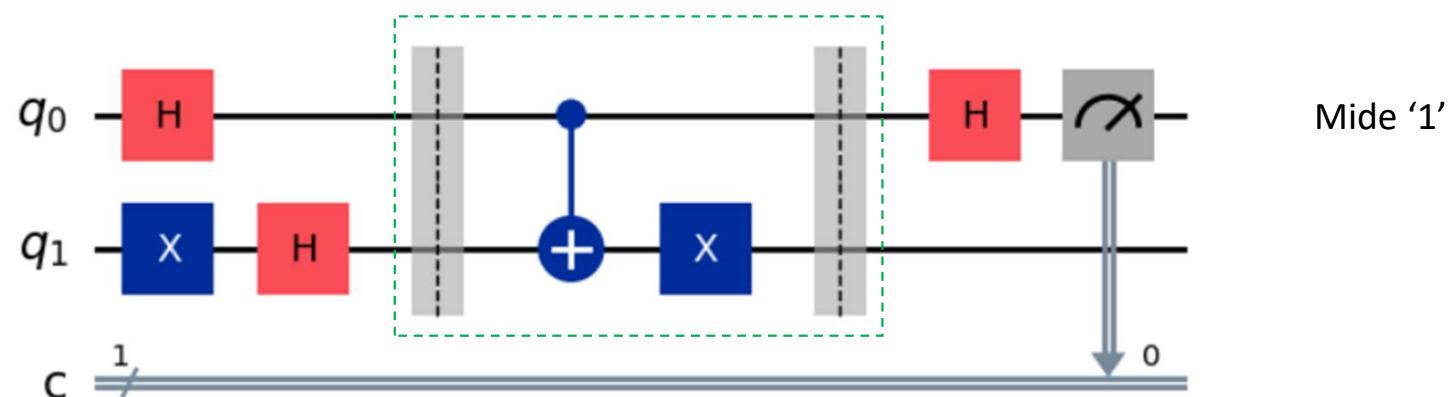
Which function? (0/1/2/3)

0: ZERO

1: ONE

2: SAME\_AS

3: OPPPOSITE\_OF  
> 3



```
{'1': 1}  
SimpleBinary.OPPOSITE_OF  
{'1': 1}  
Balanced
```

179



CamScanner