Running Qiskit 2.x in Google Colab

Package Installation and a Bell-State Example

Based on `Qiskit2_Bell_State.ipynb`

Which Packages Should Be Installed in Google Colab?

Two Main Packages Used in the Notebook

- **Qiskit with visualization extras**: supports quantum-circuit drawing and visualization.
- **Qiskit Aer**: provides high-performance local quantum-circuit simulators.

```
!pip install qiskit[visualization]
!pip install qiskit-aer
```

Why Use `[visualization]`?

It installs optional dependencies required by visualization features such as `qc.draw('mpl')` and other Qiskit plotting functions.

Installing the Packages in Colab

```
!pip install qiskit[visualization]
!pip install qiskit-aer
```

The installation also brings in optional dependencies such as `pylatexenc`, which is used by Matplotlib-based circuit drawing.

Practical Note

If Colab still reports a missing optional library immediately after installation, restart the runtime/session and execute the notebook again from the beginning.

Checking the Installed Versions

```
import qiskit
import qiskit_aer

print("Qiskit:", qiskit.__version__)
print("Qiskit Aer:", qiskit_aer.__version__)
```

Versions Recorded in the Notebook

```
Qiskit:  2.5.2
Qiskit Aer:  0.17.2
```

Because APIs can differ significantly between major Qiskit versions, checking the installed versions helps avoid incompatible code.

Bell State

Qubit Ordering Used in This Presentation

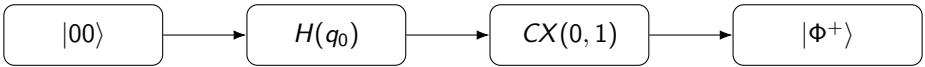
We consistently use Qiskit's displayed basis-state ordering: higher-index qubits are written on the left. For three qubits, the order is $|q_2 q_1 q_0\rangle$, so q_0 is the rightmost bit.

The two-qubit Bell state used in this example is

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Starting from $|00\rangle$, it can be generated as

$$|00\rangle \xrightarrow{H \text{ on } q_0} \frac{|00\rangle + |01\rangle}{\sqrt{2}} \xrightarrow{\mathrm{CX}_{0\to 1}} \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$



Step 1: Create the Bell-State Circuit

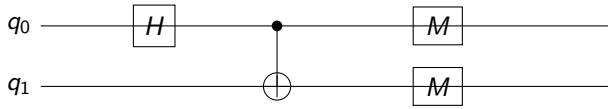
```
from qiskit import QuantumCircuit
from qiskit.visualization import plot_histogram

# 2 qubits and 2 classical bits
qc = QuantumCircuit(2, 2)

# Create a Bell state
qc.h(0)
qc.cx(0, 1)

# Measurement
qc.measure([0, 1], [0, 1])
```

Circuit Structure

H places q_0 in a quantum superposition; controlled-X entangles q_0 and q_1; finally, both qubits are measured and stored in two classical bits.

Step 2: Draw the Quantum Circuit

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



A Common Colab Issue

The `'mpl'` drawer requires optional visualization dependencies. Installing `qiskit[visualization]` avoids installing these dependencies one by one.

Step 3: Create an Aer Simulator and Transpile

```
from qiskit import transpile
from qiskit_aer import AerSimulator

simulator = AerSimulator()
qc_transpiled = transpile(qc, simulator)
```

Why Transpile?

`transpile()` transforms and optimizes a quantum circuit for a target backend. Here, the target backend is `AerSimulator`.

Conceptually,

$$\text{QuantumCircuit} \longrightarrow \text{transpile} \longrightarrow \text{backend-compatible circuit.}$$

Step 4: Run the Quantum Circuit

```
job = simulator.run(qc_transpiled, shots=1000)
result = job.result()
counts = result.get_counts()
print(counts)
```

One Actual Result Recorded in the Notebook

{'11': 513, '00': 487}

For an ideal Bell state, only 00 and 11 occur, each with theoretical probability 1/2. With a finite number of shots, their counts fluctuate statistically around 500.

Step 5: Plot the Measurement Counts

```
display(plot_histogram(counts))
```

With `shots=1000`, the histogram shows the occurrence counts of the two correlated measurement outcomes.

Expected Result

$$P(00) \approx 0.5, \qquad P(11) \approx 0.5,$$

while

$$P(01) \approx P(10) \approx 0.$$

This strong correlation in computational-basis measurements is an important feature of the Bell state.

Counts and Probabilities

The notebook explicitly normalizes counts into probabilities:

```
shots = sum(counts.values())
probabilities = {k: v / shots for k, v in counts.items()}
print(probabilities)
display(plot_histogram(probabilities))
```

Notebook Output

{'11': 0.513, '00': 0.487}

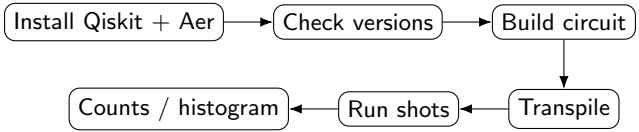
Meaning

`counts`: number of occurrences of each measurement outcome.
`probabilities`: normalized relative frequencies, $p_i = n_i/N$.

Complete Minimal Example

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

qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])

simulator = AerSimulator()
qc_t = transpile(qc, simulator)
result = simulator.run(qc_t, shots=1000).result()
counts = result.get_counts()

print(counts)
display(plot_histogram(counts))
```

Complete Workflow in Colab



1. Install the required packages.
2. Import Qiskit and verify the versions.
3. Build the quantum circuit and draw it if needed.
4. Transpile the circuit for the selected simulator/backend.
5. Execute multiple shots and obtain measurement results.
6. Visualize counts or normalized probabilities.

Key Takeaways

- In Colab, install `qiskit[visualization]` to obtain Qiskit together with its visualization dependencies.
- Install `qiskit-aer` separately to use `AerSimulator`.
- A Bell-state circuit is created using an H gate followed by a CX gate.
- Before executing the circuit on `AerSimulator`, use `transpile()` for the target backend.
- `get_counts()` returns measurement counts; normalize them if explicit probabilities are desired.
- Ideal Bell-state measurements are concentrated on 00 and 11.
- This presentation uses Qiskit's basis-state display ordering: for three qubits, $|q_2 q_1 q_0\rangle$.