在 Google Colab 中執行 Qiskit 2.x

套件安裝與 Bell State（Bell 態）範例

根據 `Qiskit2_Bell_State.ipynb`

Google Colab 中需要安裝哪些套件？

Notebook 使用的兩個主要套件

- **Qiskit（含 visualization 額外套件）**：支援量子線路繪製與視覺化。
- **Qiskit Aer**：在本機執行的高效能量子線路模擬器。

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

為何要使用 [visualization]？

它會一併安裝視覺化所需的選用相依套件，例如 `qc.draw('mpl')` 與 Qiskit 各種繪圖函式所需要的套件。

在 Colab 中安裝套件

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

上傳的 Notebook 在安裝過程中也會安裝 `pylatexenc` 等相依套件；Matplotlib 的量子線路繪圖功能會使用它。

實務注意事項

如果剛安裝完套件後，Colab 仍顯示缺少選用函式庫，可以重新啟動執行階段（runtime/session），再由 Notebook 開頭重新執行。

檢查已安裝的版本

```
import qiskit
import qiskit_aer

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

Notebook 中記錄的版本

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

由於 Qiskit 不同主要版本之間的 API 有明顯差異，因此先確認版本有助於避免使用到不相容的程式碼。

Bell State（Bell 態）

本投影片的量子位元排序

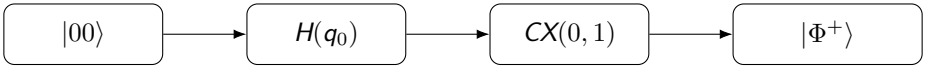
統一採用 Qiskit 的顯示順序：較高編號量子位元寫在左側。對三個量子位元使用 $|q_2 q_1 q_0\rangle$，因此 q_0 寫在最右邊。

本範例使用的二量子位元 Bell state 為

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

它可由 $|00\rangle$ 依序執行下列量子閘產生：

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



步驟 1：建立 Bell State 量子線路

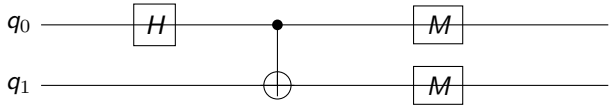
```python
from qiskit import QuantumCircuit
from qiskit.visualization import plot_histogram

# 2 個量子位元與 2 個古典位元
qc = QuantumCircuit(2, 2)

# 建立 Bell state
qc.h(0)
qc.cx(0, 1)

# 測量
qc.measure([0, 1], [0, 1])
```

量子線路結構

H 使 q_0 形成量子疊加；controlled-X 使 q_0 與 q_1 形成量子糾纏；最後測量並將結果存入兩個古典位元。

步驟 2：繪製量子線路

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



Colab 常見問題

`'mpl'` 繪圖器需要額外的視覺化相依套件。安裝 `qiskit[visualization]` 可避免逐一手動安裝。

步驟 3：建立 Aer 模擬器並進行 Transpile

```
from qiskit import transpile
from qiskit_aer import AerSimulator

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

為何需要 transpile？

`transpile()` 會依照目標 backend 轉換並最佳化量子線路；本例的目標 backend 為 `AerSimulator`。

概念上可表示為：

$$\text{QuantumCircuit} \longrightarrow \text{transpile} \longrightarrow \text{符合 backend 的量子線路.}$$

步驟 4：執行量子線路

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

Notebook 中一次實際執行結果

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

對理想 Bell state 而言，只會出現 00 與 11，理論機率皆為 $1/2$。當 shots 數有限時，兩者的次數會在 500 左右產生統計波動。

步驟 5：繪製測量次數直方圖

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

當 `shots=1000` 時，直方圖顯示兩個具有關聯性的測量結果之出現次數。

預期結果

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

while

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

這種測量結果的高度關聯性，是 Bell state 在計算基底下的重要特徵。

Counts 與 Probabilities

Notebook 將 counts 明確正規化為 probabilities：

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

Notebook 輸出

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

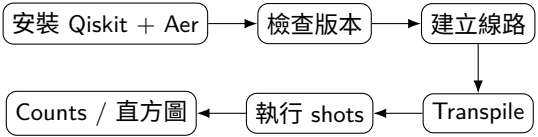
意義

`counts`：各測量結果出現的次數。
`probabilities`：正規化後的相對頻率，$p_i = n_i/N$。

完整的最小範例

```
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))
```

Colab 中的完整執行流程



1. 安裝所需套件。
2. 匯入 Qiskit 並確認版本。
3. 建立量子線路，並視需要繪製線路圖。
4. 針對所選 simulator/backend 進行 transpile。
5. 重複執行多次 shots 並取得測量結果。
6. 將 counts 或正規化後的 probabilities 視覺化。

重點整理

- 在 Colab 中安裝 `qiskit[visualization]`，即可取得 Qiskit 與繪圖所需的相依套件。
- 若要使用 `AerSimulator`，需另外安裝 `qiskit-aer`。
- Bell state 線路由 H 閘接續 CX 閘建立。
- 在 `AerSimulator` 執行量子線路前，先使用 `transpile()`。
- `get_counts()` 回傳各結果的測量次數；若需要明確的機率值，再將其正規化。
- 理想 Bell state 的測量結果會集中在 00 與 11。