前言

量子程式（quantum program）指的是可以在量子電腦（quantum computer）上執行的程式。我們正在使用的電腦以位元（bit）為基礎進行計算，而量子電腦則以量子位元（qubit）為基礎進行計算。一個位元不是 0 就是 1，但是一個量子位元則以同時是 0 又是 1 的狀態存在，只有在測量時會明確的呈現 0 或 1 的狀態。由於量子電腦的計算能力隨著量子位元數目呈現指數成長，具有目前電腦無法超越的量子霸權（quantum supremacy）特性。因而許多公司都積極致力於發展量子電腦，典型的例子有 IBM Q、Google Sycamore 以及 Rigetti 公司和 IonQ 等公司開發的量子電腦。近年來，量子電腦的量子位元數目不斷提升，這意味著量子電腦已逐漸接近實用階段。預期在不久的將來，在量子電腦上執行量子程式會成為許多人的日常。

量子電腦主要依賴量子力學中的量子疊加以及量子糾纏等現象建造，量子力學雖然發展已經超過百年，但是對於許多人而言還是一門艱澀難懂的學問。因此，許多人會認為學習量子程式設計是相當困難的。本書希望打破這個刻板印象，希望讓只有初淺程式設計經驗的讀者，即使在不懂量子力學的情況下也能夠輕鬆學習量子程式設計，寫出可以在真實量子電腦上執行的量子程式。

本書的特點是以大量的範例程式示範，讓讀者觀察執行結果。然後詳細說明每一行程式碼的設計細節與對應功能，再引導讀者學習量子電腦的計算原理與概念；並設計與實作各種著名的量子演算法，包括多伊奇 - 喬薩（Deutsch–Jozsa）演算法、格羅弗（Grover）演算法以及秀爾（Shor）演算法等。這些量子演算法都具有相當大的影響力，例如，秀爾演算法可以在量子電腦上執行，能夠在極短的時間內進行大數質因數分解，藉以破解目前使用最廣的 RSA 密碼系統。當量子電腦可用的量子位元數量足夠大而且錯誤率足夠低時，依賴 RSA 密碼系統的機制，包括線上信用卡購物、機密資料儲存與傳輸、數位簽章以及加密貨幣等將隨之瓦解，人們因而必須發展更先進的密碼系統來因應。

本書以 IBM Qiskit 為基礎，透過 IBM Quantum Lab 引導讀者輕鬆學習量子程式設計，無須安裝任何軟體，就可以直接在 IBM Q 量子電腦上執行量子程式。書中所有的原始程式碼都放置在本書專屬網頁上提供讀者下載，讓讀者可以一邊實際執行量子程式，一邊閱讀程式設計過程指引及執行結果的說明，達到「做中學」的效果。

歡迎各位讀者加入我們的行列，開始輕鬆地學習量子程式設計。

江振瑞

壬寅年于雙連坡

程式碼下載說明：

本書的範例程式碼請至書籍專屬網頁下載：*https://staff.csie.ncu.edu.tw/jrjiang/qbook/*，

1 編寫第一個量子程式

量子程式（quantum program）指的是可以在量子電腦（quantum computer）上執行的程式。我們正在使用的電腦，相對的稱為古典電腦（classical computer），以位元（bit）為基礎進行計算；而量子電腦則以量子位元（qubit）為基礎進行計算。一個位元不是 0 就是 1，但是一個量子位元則以同時是 0 又是 1 的狀態存在，只有在測量時會明確地呈現 0 或 1 的狀態。由於量子電腦的計算能力隨著量子位元數目呈現指數成長，具有古典電腦計算能力無法超越的量子霸權（quantum supremacy）特性，因此世界各國都致力於發展量子電腦，嘗試利用量子電腦執行量子程式解決各種古典電腦難以解決的複雜問題。

本章以 IBM Qiskit 為基礎，透過 IBM Quantum Lab 引導讀者輕鬆學習及編寫量子程式，並在 IBM Q 量子電腦或量子電腦模擬器上執行量子程式。Qiskit 是 IBM 公司開發的量子軟體開發工具組（software development kits, SDK），提供許多工具讓使用者方便地開發量子程式。由於 Qiskit 是以 Python 程式語言為基礎而發展的套件，透過編寫 Python 程式可以非常方便地使用，因此本書預設讀者已經具有 Python 語言程式設計的基礎。不過，若讀者不熟悉 Python 語言甚至於尚未接觸過 Python 語言也沒有關係，建議讀者可以先閱讀附錄一，以獲得 Python 程式語言最基本的資訊，快速從頭學習或複習 Python 語言。

1.1 開始編寫量子程式

我們正在使用的古典電腦發展已經相當成熟，可以很有效率的處理我們日常生活的各種事務及應用，包括文書編輯、上網瀏覽、觀看影片等。實際上，古典電腦也可以協助我們編輯量子程式送到量子電腦上執行，然後取得量子電腦的執行結果並顯示出來。以下我們引導讀者在古典電腦上使用瀏覽器取得 IBM Quantum Lab 提供

的雲端服務，獲取伺服器虛擬機（virtual machine）的計算資源，然後使用 Python 語言一步一步編寫可以在 IBM 量子電腦或是量子電腦模擬器上執行的量子程式。

IBM Quantum Lab 提供類似著名的 Jupyter Notebook 介面，讓使用者透過瀏覽器即可編寫及執行 Python 程式。因此，讀者只要在瀏覽器上開啟 IBM Quantum Lab 網頁（網址：*https://quantum-computing.ibm.com/lab*），就能夠使用 Python 語言，以 IBM Qiskit 為基礎進行量子程式設計。建議讀者註冊一個 IBMid 帳號，這樣所有在 Quantum Lab 編寫的程式可以隨時自動儲存在 IBM 雲端系統中，而且所編寫的量子程式也可以透過這個帳號很容易地在真正的 IBM 量子電腦上執行。

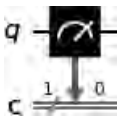
以下的範例程式是一個以 Python 語言編寫的量子程式，可以顯示 "Hello, Qubit!"，建構並顯示一個具有一個量子位元（quantum bit or qubit）以及一個傳統古典位元（classical bit）的量子線路（quantum circuit）。這個程式還不需要使用到量子電腦，它只是透過古典電腦，例如桌機、筆電或是 IBM Quantum Lab 提供的古典電腦虛擬機，執行顯示出量子線路的動作而已。除了 IBM Quantum Lab 的雲端服務之外，你也可以透過其他方式，例如 Google Colab 雲端服務或是你自己個人電腦的 Python 語言執行環境來執行這個程式。只是 IBM Quantum Lab 執行環境已經事先包含 IBM Qiskit 及相關套件，若讀者使用其他執行環境則需要另外安裝 IBM Qiskit 及相關套件。在本章最後將說明如何在 IBM Quantum Lab 以外的環境下安裝 IBM Qiskit 及相關套件。

In [1]:

```
#Program 1.1 The first quantum program
from qiskit import QuantumCircuit
print("Hello, Qubit!")
qc = QuantumCircuit(1,1)
qc.measure([0], [0])
print("This is a quantum circuit of 1 qubit and 1 bit:")
qc.draw('mpl')
```

```
Hello, Qubit!
This is a quantum circuit of 1 qubit and 1 bit:
```

```
Out[1]:
```



上列的程式碼說明如下：

- 第 1 行為程式編號及註解。
- 第 2 行使用 import 敘述引入 qiskit 套件中的 QuantumCircuit 類別。
- 第 3 行使用 print 函數顯示 "Hello, Qubit!" 字串。
- 第 4 行使用 QuantumCircuit(1,1) 建構一個包含 1 個量子位元及一個古典位元的量子線路物件，儲存於 qc 變數中。
- 第 5 行使用 QuantumCircuit 類別的 measure 方法在量子線路中加入測量單元，傳入兩個串列參數 [0] 及 [0]，以測量索引值為 0 的量子位元，並將測量結果儲存於索引值為 0 的古典位元。
- 第 6 行使用 print 函數顯示 "This is a quantum circuit of 1 qubit and 1 bit:" 字串。
- 第 7 行使用 qc.draw('mpl') 呼叫 QuantumCircuit 類別的 draw 方法，並帶入參數 'mpl'，代表透過 matplotlib 套件顯示 qc 量子線路物件對應的量子線路。請注意，qc 量子線路物件對應的量子線路簡稱 "qc 量子線路 " 或 " 量子線路 qc"，若在程式中只有唯一一個量子線路，則也直接簡稱為 " 量子線路 "。另外請注意，Python 語言中可以使用成對的雙引號表示字串，也可以使用成對的單引號表示字串，本書則依照不同狀況混合採用兩種方式表示字串。在這個量子線路中，q 代表量子位元，c 代表古典位元，1 代表古典位元的數目，而 0 代表量子位元的測量結果將儲存到索引值為 0 的古典位元。

1.2 設計量子線路

上一節的範例程式建構並顯示一個具有一個量子位元以及一個傳統古典位元的量子線路，並加入測量單元以測量量子位元的狀態儲存於古典位元中。實際上，我們可以建置具有任意數目量子位元及傳統位元的量子線路，並可針對其中特定的量

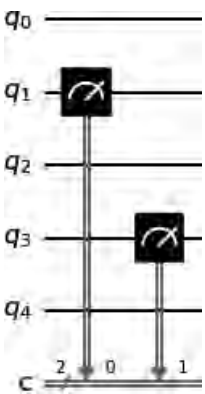
子位元進行測量儲存於指定的古典位元。這些量子線路後續都可以在量子電腦或量子電腦模擬器上執行。以下的範例程式建構並顯示一個具有 5 個量子位元以及 2 個古典位元的量子線路，其中 2 個量子位元另加上測量單元。

In [2]:

```
1 #Program 1.2 Design a quantum circuit with 5 qubits and 3 classical bits
2 from qiskit import QuantumCircuit
3 print("This is a quantum circuit of 5 qubits and 2 bits:")
4 qc = QuantumCircuit(5, 2)
5 qc.measure([1,3], [0,1])
6 qc.draw('mpl')
```

```
This is a quantum circuit of 5 qubits and 2 bits:
```

Out[2]:



上列的程式碼說明如下：

- 第 1 行為程式編號及註解。
- 第 2 行使用 import 敘述引入 qiskit 套件中的 QuantumCircuit 類別。
- 第 3 行使用 print 函數顯示 "This is a quantum circuit of 5 qubits and 2 bits:" 字串。
- 第 4 行使用 QuantumCircuit(5,2) 建構一個包含 5 個量子位元及 2 個古典位元的量子線路物件，儲存於 qc 變數中。

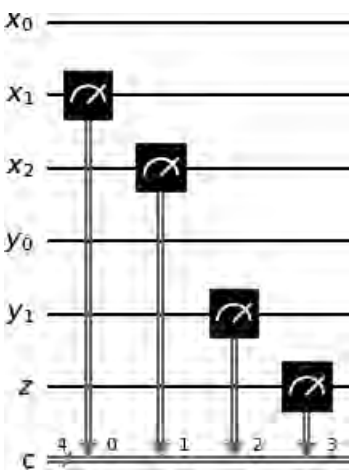
- 第 5 行使用 QuantumCircuit 類別的 measure 方法在量子線路中加入測量單元，傳入兩個串列參數 [1,3] 及 [0,1]，以測量索引值為 1 及 3 的量子位元，並分別將測量結果儲存於索引值為 0 及 1 的古典位元。
- 第 6 行使用 qc.draw('mpl') 呼叫 QuantumCircuit 類別的 draw 方法，並帶入參數 'mpl'，代表透過 matplotlib 套件顯示 qc 量子線路。量子線路中的 $q_0...q_4$ 代表索引值為 0 到 4 的量子位元，c 代表古典位元，2 代表古典位元的數目，而 0 與 1 則代表測量結果儲存到索引值為 0 與 1 的古典位元。

以下的範例程式，可以替量子線路中的量子位元以及古典位元分別命名，並加上不同的顯示標籤，讓量子線路更容易被了解。

In [3]:

```
 1 #Program 1.3 Name and label quantum bits and classical bits
 2 from qiskit import QuantumRegister,ClassicalRegister,QuantumCircuit
 3 qrx = QuantumRegister(3,'x')
 4 qry = QuantumRegister(2,'y')
 5 qrz = QuantumRegister(1,'z')
 6 cr = ClassicalRegister(4,'c')
 7 qc = QuantumCircuit(qrx,qry,qrz,cr)
 8 qc.measure([qrx[1],qrx[2]], [cr[0],cr[1]])
 9 qc.measure([4,5], [2,3])
10 qc.draw('mpl')
```

Out[3]:



上列的程式碼說明如下：

- 第 1 行為程式編號及註解。
- 第 2 行使用 import 敘述引入 qiskit 套件中的 QuantumRegister、ClassicalRegister 與 QuantumCircuit 類別。
- 第 3 行使用 qrx=QuantumRegister(3,'x') 建構一個包含 3 個量子位元的量子暫存器物件，顯示標籤為 'x'，儲存於 qrx 變數中，這 3 個位元在 qrx 的區域索引值為 0、1、2，全域索引值為 0、1、2。
- 第 4 行使用 qry=QuantumRegister(2,'y') 建構一個包含 2 個量子位元的量子暫存器物件，顯示標籤為 'y'，儲存於 qry 變數中，這 2 個位元在 qry 的區域索引值為 0、1，全域索引值為 3、4。
- 第 5 行使用 qry=QuantumRegister(1,'z') 建構一個包含 1 個量子位元的量子暫存器物件，顯示標籤為 'z'，儲存於 qrz 變數中，這 1 個位元在 qrz 的區域索引值為 0，全域索引值為 5。
- 第 6 行使用 cr=ClassicalRegister(4,'c') 建構一個包含 4 個古典位元的古典暫存器物件，顯示標籤為 'c' 以代表儲存量子位元測量的古典位元，儲存於 cr 變數中，這 4 個位元在 cr 的區域索引值為 0、1、2、3，全域索引值為 0、1、2、3。
- 第 7 行使用 qc=QuantumCircuit(qrx,qry,qrz,cr) 建構一個包含量子暫存器物件 qrx 的 3 個量子位元、量子暫存器物件 qry 的 2 個量子位元、量子暫存器物件 qrz 的 1 個量子位元，以及古典暫存器物件 cr 的 4 個古典位元的量子線路物件，儲存於 qc 變數中。
- 第 8 行使用 qc.measure([qrx[1],qrx[2]],[cr[0],cr[1]]) 呼叫 QuantumCircuit 類別的 measure 方法，測量量子暫存器物件 qrx 的 qrx 區域索引值為 1 與 2 的量子位元，並將測量結果儲存於古典暫存器物件 cr 的 cr 區域索引值為 0 與 1 的古典位元。
- 第 9 行使用 qc.measure([4,5],[2,3]) 呼叫 QuantumCircuit 類別的 measure 方法，測量量子暫存器中全域索引值為 4 與 5 的量子位元，並將測量結果儲存於古典暫存器中全域索引值為 2 與 3 的古典位元。

- 第 10 行使用 qc.draw('mpl') 呼叫 QuantumCircuit 類別的 draw 方法，並帶入參數 'mpl'，代表透過 matplotlib 套件顯示量子線路。

目前我們編寫的程式都只是在古典電腦上建構並顯示量子線路而已，在下一節中，我們展示如何在 IBM 量子電腦模擬器 AerSimulator 上執行量子程式中的量子線路，也簡稱執行量子線路或執行量子程式。我們需要用到 transpile 函數將量子線路轉譯（transpile）為 OpenQASM 碼，然後藉以在後端（backend）量子電腦模擬器（對應 AerSimulator 類別）上執行，得到測量結果並儲存於古典位元上，最後再以古典電腦計算並顯示多次執行的統計結果。OpenQASM 是開放量子組合語言（Open Quantum Assembly Language），是一種量子指令的中間型式表示（intermediate representation）語言。該語言在 2017 年 7 月首度發表並使用在 IBM 的 Qiskit 套件中，在 Qiskit 套件中也簡稱為 QASM。

1.3 使用量子電腦模擬器執行量子程式

本節說明如何使用量子電腦模擬器執行量子程式。首先，以下的範例程式展示透過轉譯（transpile）的方式在量子電腦模擬器上執行量子程式：

In [4]:

```
#Program 1.4 Transpile and execute quantum circuit on simulator
from qiskit import QuantumCircuit, transpile, execute
from qiskit.providers.aer import AerSimulator
sim = AerSimulator()
qc = QuantumCircuit(1, 1)
qc.measure([0], [0])
print(qc)
cqc = transpile(qc, sim)
job=execute(cqc, backend=sim, shots=1000)
result = job.result()
counts = result.get_counts(qc)
print("Total counts for qubit states are:",counts)
```

```
     ┌─┐
  q: ┤M├
     └╥┘
c: 1/═╩═
      0
Total counts for qubit states are: {'0': 1000}
```

上列的程式碼說明如下：

- 第 1 行為程式編號及註解。
- 第 2 行使用 import 敘述引入 qiskit 套件中的 QuantumCircuit 類別、transpile 函數以及 execute 函數。
- 第 3 行使用 import 敘述引入 qiskit.providers.aer 中的 AerSimulator 類別。
- 第 4 行使用 AerSimulator() 建構 IBM QASM 量子電腦模擬器物件，儲存於 sim 變數中。
- 第 5 行使用 QuantumCircuit(1,1) 建構一個包含 1 個量子位元及一個古典位元的量子線路物件，儲存於 qc 變數中。
- 第 6 行使用 QuantumCircuit 類別的 measure 方法在量子線路中加入測量單元，傳入兩個串列參數 [0] 及 [0]，以測量索引值為 0 的量子位元，並將測量結果儲存於索引值為 0 的古典位元。
- 第 7 行使用 print(qc) 呼叫 print 函數以文字模式顯示 qc 量子線路。請注意，此處我們不使用 qc.draw('mpl') 顯示量子線路，這是因為在這行之後還有呼叫 print 函數顯示文字內容的敘述，這會造成 qc.draw('mpl') 敘述無法正確顯示量子線路，因此我們改用 print(qc) 敘述顯示量子線路。請注意，若使用 Jupyter Notebook 執行環境所提供的 display 函數，可以解決這個問題。具體的說，在本行可以使用 display(qc.draw('mpl')) 正確的以 matplotlib 模式顯示量子線路，我們在本章後面的其他章節中有時後會採取這種作法。
- 第 8 行使用 transpile 函數將 qc 量子線路轉譯（transpile）為量子電腦模擬器可以執行的 OpenQASM 指令，儲存於 cqc 變數中。
- 第 9 行呼叫 execute 函數建立一個工作，儲存於 job 變數中，其中傳入參數 cqc 表示要執行的 OpenQASM 指令區塊，backend=sim 設定在後端使用 sim 物件所指定的量子電腦模擬器，shots=1000 設定在後端量子電腦模擬器上執行 OpenQASM 指令區塊 1000 次（請注意，若 shots 參數未指定則其預設值為 1024），而每次執行都測量量子位元並將測量結果儲存於古典位元中。
- 第 10 行使用 job 物件的 result 方法取得 job 物件的執行相關資訊，儲存於物件變數 result 中。執行相關資訊除了執行環境之外，也包括執行結果，也就是量子線路在量子電腦模擬器上的執行結果。

- 第 11 行使用 result 物件的 get_counts(qc) 方法取出有關 qc 量子線路測量結果的計數（counts），並以字典（dict）型別儲存於變數 counts 中。
- 第 12 行使用 print 函數顯示 "Total counts for qubit states are :" 字串及字典型別變數 counts 的值，在這個程式中 counts 變數的值為 {'0': 1000}，也就是測量結果為 '0' 的計數為 1000 次。

上列範例程式將量子線路轉譯為 OpenQASM 碼，然後在後端量子電腦模擬器上執行這個量子線路 1000 次，最後測量量子線路中唯一一個量子位元的狀態並儲存於古典位元上。由於量子位元的預設初始狀態為狀態 '0'，因此執行量子線路 1000 次的測量結果都是狀態 '0'。請注意，在 Qiskit 套件中使用包含一個或多個字元 '0' 與字元 '1' 的字串代表量子位元狀態，這是一個非常方便表示量子位元狀態的表示法。

前一個範例程式將量子線路轉譯之後在量子電腦模擬器上執行。實際上，我們不需要執行轉譯動作也可以在電腦模擬器上執行量子線路。這是因為即使我們在量子程式中沒有明確列出轉譯步驟，系統還是會自動進行轉譯之後才執行量子線路。以下的範例程式就是不轉譯量子線路，而是以量子電腦模擬器直接執行量子線路。這個範例程式與前一個範例程式非常相似，它與前一個範例程式相比，只是少了轉譯步驟而已，也就是少了 cqc=transpile(qc,sim) 敘述，然後在呼叫 execute 函數時傳入參數 qc，而不是參數 cqc，表示要執行的就是原始的 qc 量子線路本身。

In [5]:

```
 1 #Program 1.5 Execute quantum circuit (program) on simulator
 2 from qiskit import QuantumCircuit, execute
 3 from qiskit.providers.aer import AerSimulator
 4 sim = AerSimulator()
 5 qc = QuantumCircuit(1, 1)
 6 qc.measure([0], [0])
 7 print(qc)
 8 job=execute(qc, backend=sim, shots=1000)
 9 result = job.result()
10 counts = result.get_counts(qc)
11 print("Total counts for qubit states are:",counts)
```

```
  q: ┤M├
c: 1/╚═╝
     0
```

```
Total counts for qubit states are: {'0': 1000}
```

上列的程式碼說明如下：

- 第 1 行為程式編號及註解。
- 第 2 行使用 import 敘述引入 qiskit 套件中的 QuantumCircuit 類別以及 execute 函數。
- 第 3 行使用 import 敘述引入 qiskit.providers.aer 中的 AerSimulator 類別。
- 第 4 行使用 AerSimulator() 建構 IBM QASM 量子電腦模擬器物件，儲存於 sim 變數中。
- 第 5 行使用 QuantumCircuit(1,1) 建構一個包含 1 個量子位元及一個古典位元的量子線路物件，儲存於 qc 變數中。
- 第 6 行使用 QuantumCircuit 類別的 measure 方法在量子線路中加入測量單元，傳入兩個串列參數 [0] 及 [0]，以測量索引值為 0 的量子位元，並將測量結果儲存於索引值為 0 的古典位元。
- 第 7 行使用 print(qc) 呼叫 print 函數以文字模式顯示量子線路。
- 第 8 行呼叫 execute 函數建立一個工作，儲存於 job 變數中，其中傳入參數 qc 表示要執行的量子線路，backend=sim 設定在後端使用 sim 物件所指定的量子電腦模擬器，shots=1000 設定在後端量子電腦模擬器上執行量子線路 qc 共 1000 次，而每次執行都測量量子位元並將測量結果儲存於古典位元中。
- 第 9 行使用 job 物件的 result 方法取得 job 物件的執行相關資訊，儲存於物件變數 result 中。執行相關資訊除了執行環境之外，也包括執行結果，也就是量子線路在量子電腦模擬器上的執行結果。
- 第 10 行使用 result 物件的 get_counts(qc) 方法取出有關 qc 量子線路量測結果的計數（counts），並以字典（dict）型別儲存於變數 counts 中。
- 第 11 行使用 print 函數顯示 "Total counts for qubit states are :" 字串及字典型別變數 counts 的值，在這個程式中 counts 變數的值為 {'0': 1000}，也就是測量結果為 '0' 的計數為 1000 次。

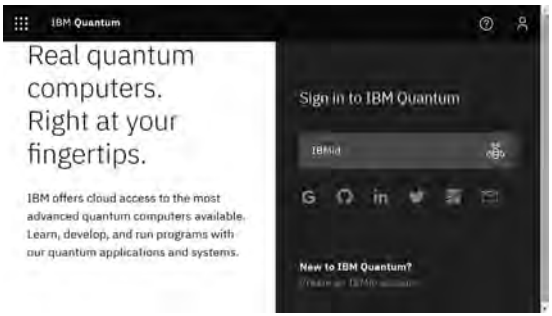
上列兩個範例程式透過量子電腦模擬器執行量子程式，在下節中，我們展示如何透過網路連線到真實的 IBM Q 量子電腦執行量子程式。

1.4　使用量子電腦執行量子程式

本節介紹如何在 IBM Q 量子電腦上執行量子程式。首先，讀者要取得存取 IBM Q 系統的 token，然後進行儲存 token 與載入 token 的動作，說明如下：

- 登入 IBM Quantum 系統（簡稱 IBM Q 系統）：

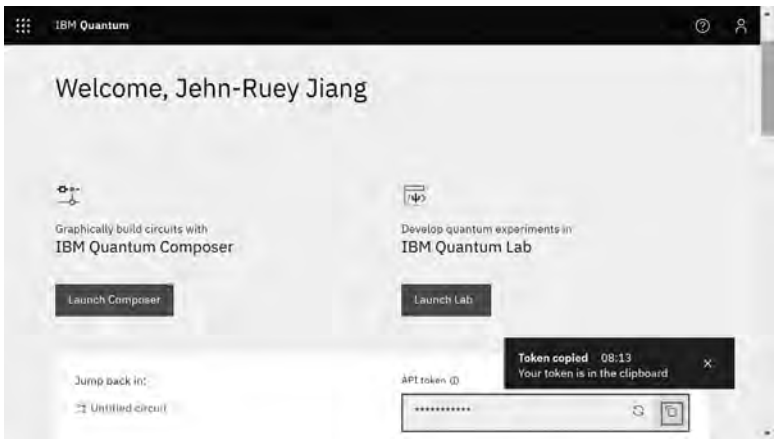
  使用原有帳號登入 IBM Q 系統（網址：*https://quantum-computing.ibm.com/login*），或註冊一個新帳號後登入系統，其網頁畫面如下所示：



（圖片來源：IBM Quantum 網站畫面）

- 取得 IBM Q 系統 token：

  登入 IBM Q 系統之後就可以取得 IBM Q 系統的 token。如下圖所示，按下右下方正方形區域內的複製圖標就可以將 IBM Q 系統的 token 複製到剪貼簿中。實際上，token 是一個包含 128 字元的字串。



（圖片來源：IBM Quantum 網站畫面）

- 儲存 IBM Q 系統 token：

  在量子程式中使用 qiskit 套件 IBMQ 類別的 save_account 方法將 token 儲存在本地儲存空間中。若讀者使用 IBM Quatum Lab，則 token 是儲存在 Quatum Lab 執行環境中，其用法為：

  IBMQ.save_account('.....')，其中 ..... 代表填入在上一個步驟取得的 IBM Q 系統的 token，也就是 128 個字元的字串。

  請注意，儲存 IBM Q 系統 token 的動作只需要進行一次即可。但是，若讀者重新產生新的 token，則必須再重新執行儲存 IBM Q 系統 toke 的動作一次。這時要使用的方法為：

  IBMQ.save_account('.....',overwrite=True)，其中 ..... 代表填入重新產生的 IBM Q 系統 token，而 overwrite=True 代表要覆蓋原來儲存的舊 token 內容。

- 載入 IBM Q 系統 token：

  在量子程式中使用 IBMQ 類別 load_account 方法將前一個步驟儲存的 token 載入，就可以開始使用真實的 IBM Q 量子電腦執行量子程式。其用法為：

  IBMQ.load_account()

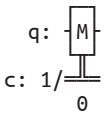
  請注意，每次重新連線到 IBM Quantum Lab 環境中，都需要進行 IBM Q 系統 token 的載入動作一次，也就是執行 IBMQ.load_account() 一次。但是連線之後在第二次執行 IBMQ.load_account() 敘述時會引發 "Credentials are already in use. The existing account in the session will be replaced." 的警告訊息，此時可以不用理會這個訊息。

在完成 IBM Q 系統 token 的取得、儲存及載入之後，就可以在 IBM Q 量子電腦上執行量子程式了。以下的範例程式可以連線到 IBM Q 量子電腦，在實際的量子電腦上執行：

In [6]:

```
1 #Program 1.6 Execute quantum circuit (program) on least busy quantum computer
2 from qiskit import QuantumCircuit, IBMQ, execute
3 from qiskit.providers.ibmq import least_busy
4 from qiskit.tools.monitor import job_monitor
5 qc = QuantumCircuit(1, 1)
```

```
 6 qc.measure([0], [0])
 7 print(qc)
 8 #IBMQ.save_account('......',overwrite=True)
 9 IBMQ.load_account()
10 provider=IBMQ.get_provider(group='open')
11 print(provider)
12 qcomp = least_busy(provider.backends(simulator=False))
13 print("The least busy quantum computer is:",qcomp)
14 job=execute(qc, backend=qcomp, shots=1000)
15 job_monitor(job)
16 result = job.result()
17 counts = result.get_counts(qc)
18 print("Total counts for qubit states are:",counts)
```




```
ibmqfactory.load_account:WARNING:2022-06-08 01:22:00,786: Credentials are already
in use. The existing account in the session will be replaced.

<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
The least busy quantum computer is: ibmq_armonk
Job Status: job has successfully run
Total counts for qubit states are: {'0': 984, '1': 16}
```

上列的程式碼說明如下：

- 第 1 行為程式編號及註解。
- 第 2 行使用 import 敘述引入 qiskit 套件中的 QuantumCircuit 及 IBMQ 類別以及 execute 函數。
- 第 3 行使用 import 敘述引入 qiskit.providers.ibmq 中的 least_busy 函數。
- 第 4 行使用 import 敘述引入 qiskit.tools.monitor 中的 job_monitor 函數。
- 第 5 行使用 QuantumCircuit(1,1) 建構一個包含 1 個量子位元及一個古典位元的量子線路物件，儲存於 qc 變數中。
- 第 6 行使用 QuantumCircuit 類別的 measure 方法在量子線路中加入測量單元，傳入兩個串列參數 [0] 及 [0]，以測量索引值為 0 的量子位元，並將測量結果儲存於索引值為 0 的古典位元。

- 第 7 行使用 print(qc) 呼叫 print 函數以文字模式顯示量子線路。
- 第 8 行使用 IBMQ.save_account('.....',overwrite=True) 方法，將 token 存到區域檔案系統中，其中 ..... 代表 token 字串，而 overwrite=True 代表要覆蓋原來儲存的舊 token 內容。請注意，token 儲存的動作只要執行一次，在 token 已經儲存之後只要載入這個 token 就可以使用 token 的權限執行量子程式。同樣的，若讀者使用 IBM Quantum Lab，也只要執行一次儲存 token 字串的動作一次，IBM Quantum Lab 會自動將 token 儲存在環境設定檔案 qiskitrc 中，提供後續載入使用。請注意，此行被標註為註解是因為 token 儲存只需要進行一次，而且其中包含 128 字元的 token 因為保密的關係已經以 ..... 取代了。
- 第 9 行使用 IBMQ.load_account() 方法載入儲存在區域檔案系統或是 IBM Quantum Lab 系統中環境設定檔案的 token。與 token 儲存不同的是，token 載入的動作在每一次重新連線至 IBM Quantum Lab 系統時都需要再執行一次。
- 第 10 行使用 IBMQ.get_provider(group='open') 方法以 group 名稱為 'open' 的條件取得 IBM Q 的設備提供服務物件，儲存於物件變數 provider 中。
- 第 11 行使用 print 函數顯示 IBM Q 的設備提供服務物件變數 provider 的相關資訊。
- 第 12 行使用 qcomp=least_busy(provider.backends(simulator=False)) 先呼叫 provider 物件的 backends 方法，設定 simulator=False 條件，以取得後端實際的量子電腦。然後呼叫 least_busy 函數選出負載最小最不忙碌（least busy）的量子電腦，儲存在物件變數 qcomp 中。
- 第 13 行使用 print 函數顯示 "The least busy quantum computer is:" 訊息，並在訊息之後顯示 qcomp 變數所儲存的實際量子電腦資訊。
- 第 14 行使用 execute(qc, backend=qcomp, shots=1000) 函數建立一個工作，儲存於 job 變數中。backend=qcomp 設定在後端使用 qcom 物件所指定的實際量子電腦執行工作，並透過 shots=1000 設定工作一共執行 1000 次（shot），每次都測量量子位元並將測量結果儲存於古典位元中保存下來。請注意，目前 IBM 量子電腦支援的 shots 數量預設為 1024，而最高為 20000。

- 第 15 行使用 job_monitor(job) 函數監督 job 的執行狀態。可能出現的訊息為：

  Job Status: job is being validated
  Job Status: job is queued (123)
  （上列的數字會漸次減少，表示在 queue 中的工作正一一完成中）
  Job Status: job is actively running
  Job Status: job has successfully run

- 第 16 行使用 job 物件的 result 方法取得 job 物件的執行相關資訊，儲存於變數 result 中。執行相關資訊包括執行結果，也就是量子線路在量子電腦模擬器上的執行結果。
- 第 17 行使用 get_counts(qc) 函數取出量子線路各種量測結果的計數（counts），並以字典（dict）型別儲存於變數 counts 中。
- 第 18 行使用 print 函數顯示 "Total counts for qubit states are :" 字串及字典型別變數 counts 的值，在這個程式中 counts 變數的值為 {'0': 984, '1': 16}，也就是測量結果為 '0' 的計數為 984 次，機率為 98.4%；而測量結果為 '1' 的計數為 16 次，機率為 1.6%。

上列範例程式先呼叫 least_busy 方法選出 IBM 可提供外界使用，且負載最小最不忙碌（least busy）的量子電腦，然後在這個量子電腦上執行量子程式，儲存在物件變數 qcomp 中。如上列範例程式執行結果中所顯示，這個範例程式是在名稱代號為 'ibmq_armonk' 的量子電腦上執行的。這是自動挑選出來負載最小的，但是僅具有 1 個量子位元的量子電腦。因為上列範例程式的量子線路恰好只包含 1 個量子位元，因此量子程式可以正確執行。但是，若量子程式的量子線路包含 1 個以上的量子位元，則量子程式就無法執行了。因此建議讀者參考以下的範例程式以及 IBM 網站（網址：*https://quantum-computing.ibm.com/services?services=systems&view=table*），選擇負載不大且位元數量足夠的量子電腦來執行量子程式。

In [7]:

```
1 #Program 1.7 Execute quantum circuit (program) on proper quantum computer
2 from qiskit import QuantumCircuit, IBMQ, execute
3 from qiskit.tools.monitor import job_monitor
4 qc = QuantumCircuit(1, 1)
5 qc.measure([0], [0])
```