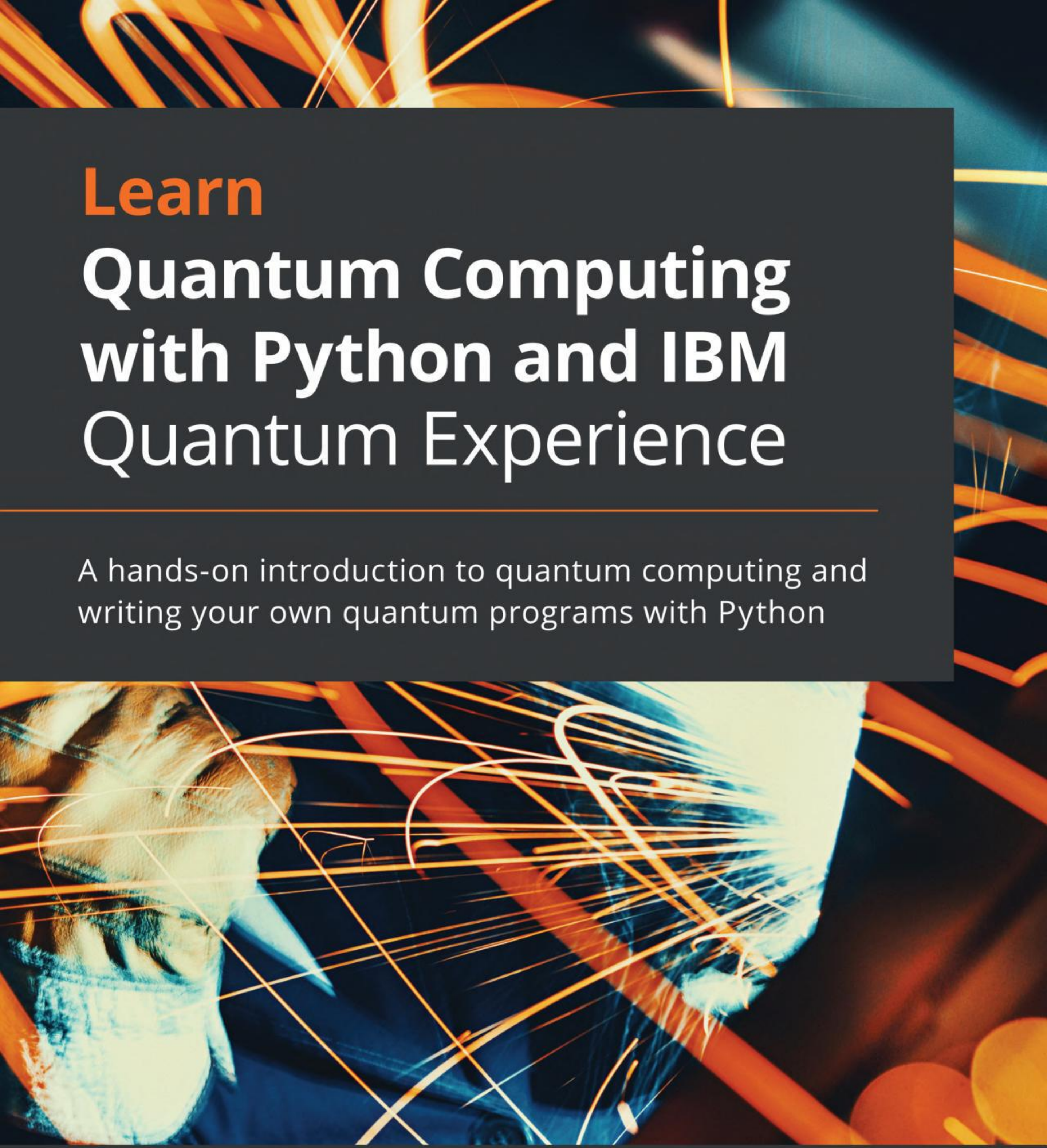
Learn Quantum Computing with Python and IBM Quantum Experience

A hands-on introduction to quantum computing and writing your own quantum programs with Python



Robert Loredo

Learn Quantum Computing with Python and IBM Quantum Experience

Commissioning Editor: Richa Tripathi
Acquisition Editor: Alok Dhuri
Senior Editor: Rohit Singh
Content Development Editor: Rosal Colaco
Technical Editor: Gaurav Gala
Copy Editor: Safis Editing
Project Coordinator: Deeksha Thakkar
Proofreader: Safis Editing
Indexer: Priyanka Dhadke
Production Designer: Aparna Bhagat

First published: September 2020

Production reference: 1250920

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-83898-100-6

`www.packt.com`

Table of Contents

3
Creating Quantum Circuits using Quantum Lab Notebooks

Section 2: Basics of Quantum Computing

4
Understanding Basic Quantum Computing Principles

5
Understanding the Quantum Bit (Qubit)

8
Programming with Qiskit Terra

9
Monitoring and Optimizing Quantum Circuits

10
Executing Circuits Using Qiskit Aer

Preface

IBM Quantum Experience is a platform that enables developers to learn the basics of quantum computing by allowing them to run experiments on a quantum computing simulator and a real device. This book will explain the basic principles of quantum computing, along with one principle of quantum mechanics, entanglement, and the implementation of quantum algorithms and experiments on IBM's quantum processors.

This book provides you with a step-by-step introduction to quantum computing using the **IBM Quantum Experience** platform. You will learn how to build quantum programs on your own, discover early use cases in your business, and help to get your company equipped with quantum computing skills.

You will start working with simple programs that illustrate quantum computing principles and slowly work your way up to more complex programs and algorithms that leverage advanced quantum computing algorithms. As you build on your knowledge, you'll understand the functionality of the IBM Quantum Experience and the various resources it offers.

We'll explore quantum computing principles such as superposition, entanglement, and interference, then we'll become familiar with the contents and layout of the IBM Quantum Experience dashboard.

Then, we'll understand quantum gates and how they operate on qubits and discover the **Quantum Information Science Kit** (**Qiskit**) and its elements such as Terra and Aer.

We'll then get to grips with quantum algorithms such as Deutsch-Jozsa, Simon, Grover, and Shor's algorithms, and then visualize how to create a quantum circuit and run the algorithms on any of the available quantum computers hosted on the IBM Quantum Experience.

Furthermore, you'll learn the differences between the various quantum computers and the different types of simulators available. Later, you'll explore the basics of quantum hardware, pulse scheduling, quantum volume, and how to analyze and optimize your quantum circuits, all while using the resources available on the IBM Quantum Experience.

By the end of this book, you'll have learned how to build quantum programs on your own and will have gained practical quantum computing skills that you can apply to your research or industry.

Who this book is for

This book is for Python developers who are interested in learning about quantum computing and expanding their abilities to solve classically intractable problems with the help of the IBM Quantum Experience and Qiskit. Some background in computer science, physics, and some linear algebra is required.

What this book covers

Chapter 1, *Exploring the IBM Quantum Experience*, will be your guide to the IBM Q Experience dashboard. This chapter will describe the layout and what each section in the dashboard means. The dashboard might alter over time, but the basic information should still be available to you.

Chapter 2, *Circuit Composer – Creating a Quantum Circuit*, will help you learn about Circuit Composer. This chapter will outline the user interface that will assist you in learning about quantum circuits, the qubits, and their gates that are used to perform operations on each qubit.

Chapter 3, *Creating Quantum Circuits Using Quantum Lab Notebooks*, will help you learn how to create circuits using the Notebook with the latest version of Qiskit already installed on the IBM Quantum Experience. You will learn how to save, import, and leverage existing circuits without having to install anything on your local machine.

Chapter 4, *Understanding Basic Quantum Computing Principles*, will help you learn about the basic quantum computing principles used by the IBM Quantum systems, particularly, superposition, entanglement, and interference. These three properties, often used together, serve as the base differentiators that separate quantum systems from classical systems.

Chapter 5, *Understanding the Quantum Bit (Qubit)*, will help you learn about the basic fundamental component of a quantum system, the quantum bit or qubit, as it is often called. After reading this chapter, you will understand the basis states of a qubit, how they are measured, and how they can be visualized both mathematically and graphically.

Chapter 6, *Understanding Quantum Logic Gates*, will help you learn how to perform operations on a qubit. These operations are often referred to as quantum gates. This chapter will enable you, via the IBM Quantum Experience, to get to grips with the operations that each of these quantum gates performs on a qubit and the results of each of those operations. Examples of the quantum principles such as reversibility, which is a core principle for all quantum gates, will be included.

Chapter 7, *Introducing Qiskit and its Elements*, will help you learn about Qiskit and all of its libraries that can help you develop and implement various quantum computing solutions. Qiskit is composed of four elements, each of which has a specific functionality and role that can be leveraged based on the areas you wish to experiment in. The elements are Terra (Earth), Aer (Air), Ignis (Fire), and Aqua (Water). This chapter will also discuss how to contribute to each of the elements and how to install it locally on your machine.

Chapter 8, *Programming with Qiskit Terra*, will help you learn about the basic foundational element, Terra. Terra is the base library upon which all the other elements of Qiskit are built. Terra allows a developer to code the base of an algorithm to the specific operator on a qubit. This is analogous to assembly language with just a slightly easier set of library functions. It will also include a section on the Pulse library, which allows you to create pulse schedules to manipulate the quantum qubits via the hardware.

Chapter 9, *Monitoring and Optimizing Quantum Circuits*, will help you learn how to monitor the job requests sent to either the simulator or the quantum computers on the IBM Quantum Experience. Optimization features will also be covered here to allow you to leverage many of the existing optimization features included in the Qiskit libraries or to create your own custom optimizers.

Chapter 10, *Executing Circuits Using Qiskit Aer*, will help you learn about Qiskit Aer, a high-performance framework that you will use to simulate your circuits on various optimized simulator backends. You will learn what the differences are between the four various simulators of Qasm, State vector, Unitary, and Pulse, and what functionality each one exhibits. Aer also contains tools you can use to construct noise models, should you need to perform some research to reproduce errors due to noise.

Chapter 11, *Mitigating Quantum Errors Using Ignis*, will help you learn about the various errors that currently affect experiments on read devices, such as relaxation and decoherence, so you can design quantum error correction codes. You will also learn about readout error mitigation, which is a way to mitigate the readout errors returned from a quantum computer.

Chapter 12, *Learning about Qiskit Aqua*, will, in essence, pull everything together so that end users such as researchers and developers from the various domains of chemistry, machine learning, finance, optimization, and more can run their computations on a quantum computer system without having to know all the inner workings. Aqua is the tool connected to quantum algorithms that has been created to do just that. You will learn how to extend your classical application to include running a quantum algorithm.

Chapter 13, *Understanding Quantum Algorithms*, will dig into some basic algorithms using the IBM Quantum Experience Composer. This chapter will start with some simple algorithms that illustrate the advantages of superposition and entanglement, such as Bell's state theorem, and extends into some more common algorithms to solve some problems that illustrate uses of superposition and entanglement such as Deutsch-Josza and a few others, each of which provides some variance to the different algorithm types.

Chapter 14, *Applying Quantum Algorithms*, describes the various quantum computing properties and algorithms used to create some of the more well-known algorithms such as Quantum Amplitude Estimation, Variational Quantum Eigensolvers, and Shor's algorithm.

Appendix A, *Resources*, will help you get familiar with all the available resources in the IBM Quantum Experience and Qiskit community. These resources that have been contributed either by the Qiskit open source community, or the IBM Quantum research teams themselves. The information is laid out so anyone with basic to expert-level knowledge can jump in and start learning. There is a full quantum course, textbook, and Slack community that you can connect to in order to extend your learning and collaborate with others.

Assessments contains the answers to the questions asked in the chapters.

To get the most out of this book

You will need to have internet access to connect to the IBM Quantum Experience. Since the IBM Quantum Experience is hosted on the IBM Cloud, you will not need anything more other than a supported browser and to register with the IBM Quantum Experience. Everything else is taken care of on the IBM Quantum Experience.

Software/hardware covered in the book	**OS requirements**
Latest browser (Firefox, Chrome, Safari)	Windows, Mac OS X, and Linux (any)

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`.
2. Select the **Support** tab.
3. Click on **Code Downloads**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Learn-Quantum-Computing-with-Python-and-IBM-Quantum-Experience`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Code in Action

Code in Action videos for this book can be viewed at `https://bit.ly/35o5M80`.

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781838981006_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "This will initialize our `t1`, `a`, and `b` parameters, which we will use to generate `T1Fitter`."

A block of code is set as follows:

```
# Initialize the parameters for the T1Fitter, A, T1, and B
param_t1 = t1*1.2
param_a = 1.0
param_b = 0.0
```

Any command-line input or output is written as follows:

```
[[1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "As shown in the following screenshot, **ibmq_qasm_simulator** can run wider circuits than most local machines and has a larger variety of basis gates."

Tips or important notes
Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

Section 1: Tour of the IBM Quantum Experience (QX)

In this section, we will tour all the features and resources available to you on the IBM Quantum Experience. These will include some educational materials for all levels, information on the many simulators and real devices available to you, and tools that you can use to perform experiments from the many tutorials as you learn, or to simply create experiments on your own.

This section comprises the following chapters:

1
Exploring the IBM Quantum Experience

Quantum computing has been growing in popularity over the past few years, most recently since IBM released the **IBM Quantum Experience** (**IQX**) back in May 2016. This release was the first of its kind, hosted on the cloud and providing the world with the opportunity to experiment with a quantum computer for free. The IQX includes a user interface that allows anyone to run experiments on both a simulator and on a real quantum computer.

The goal of this chapter is to first introduce you to the IBM Quantum Experience site, specifically the **dashboard**, which contains everything you need in order to run experiments. It also allows you to experiment with existing experiments contributed by other developers from around the world, the benefits of which can help you to understand how others are experimenting, and you can perhaps collaborate with them if the experiments correlate with your own ideas.

This chapter will help you understand what actions and information are available in each view. This includes creating an experiment, running experiments on a simulator or real quantum device, information about your profile, available backends, or pending results to experiments. So, let's get started!

The following topics will be covered in this chapter:

- Navigating the IBM Quantum Experience
- Getting started with IBM Quantum Experience

Technical requirements

Throughout this book, it is expected that you will have some experience in developing with Python and, although it isn't necessary, some basic knowledge of classical and quantum mechanics would help.

Most of the information will be provided with each chapter, so if you do not have knowledge of classical or quantum mechanics, we will cover what you need to know here.

For those of you that do have knowledge, the information here will serve as a refresher. The Python editor used throughout this book is **Jupyter Notebook**. You can, of course, use any Python editor of your choice. This may include **Watson Studio**, **PyCharm**, **Spyder**, **Visual Studio Code**, and so on. Here is the link for the CiA videos: `https://bit.ly/35o5M80`

Here is the source code used throughout this book: `https://github.com/PacktPublishing/Learn-Quantum-Computing-with-Python-and-IBM-Quantum-Experience`.

Navigating the IBM Quantum Experience

As mentioned earlier, the dashboard is your high-level view of what you will normally see once you log in to IQX. It aggregates multiple views that you can see, and this helps you to get an idea as to what machines you have access to and what experiments you have pending, running, or completed.

In this section, we will go through the steps to get registered on IQX. Let's do that in the next section.

Registering to the IBM Quantum Experience

In this section, we will get registered and explain what happens in the background once you sign up to IQX for the first time. This will help you understand what features and configurations are prepared and available to you upon registration.

To register to the IBM Quantum Experience, follow these steps:

1. The first step is to head over to the IBM Quantum Experience site at the following link: `https://quantum-computing.ibm.com/`
2. Sign-in to your account from the login screen, as shown in *Figure 1.1*. Your individual situation will determine how to proceed from there.

   If you already have an account or are already signed in, you can skip this section and move on to the next one.

   If you have not registered, then you can select the login method of your choice from the sign-in screen. As you can see, you can register using various methods, such as with your **IBM ID**, **Google**, **GitHub**, **Twitter**, **LinkedIn**, or by email.

   If you do not have any of the account types listed, then you can simply register for an **IBMid** account and use that to sign in:

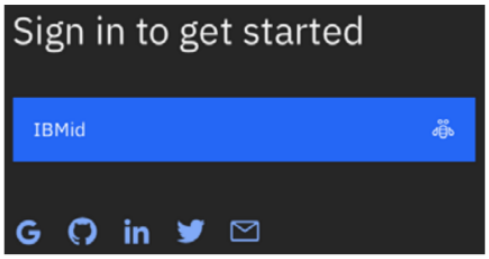
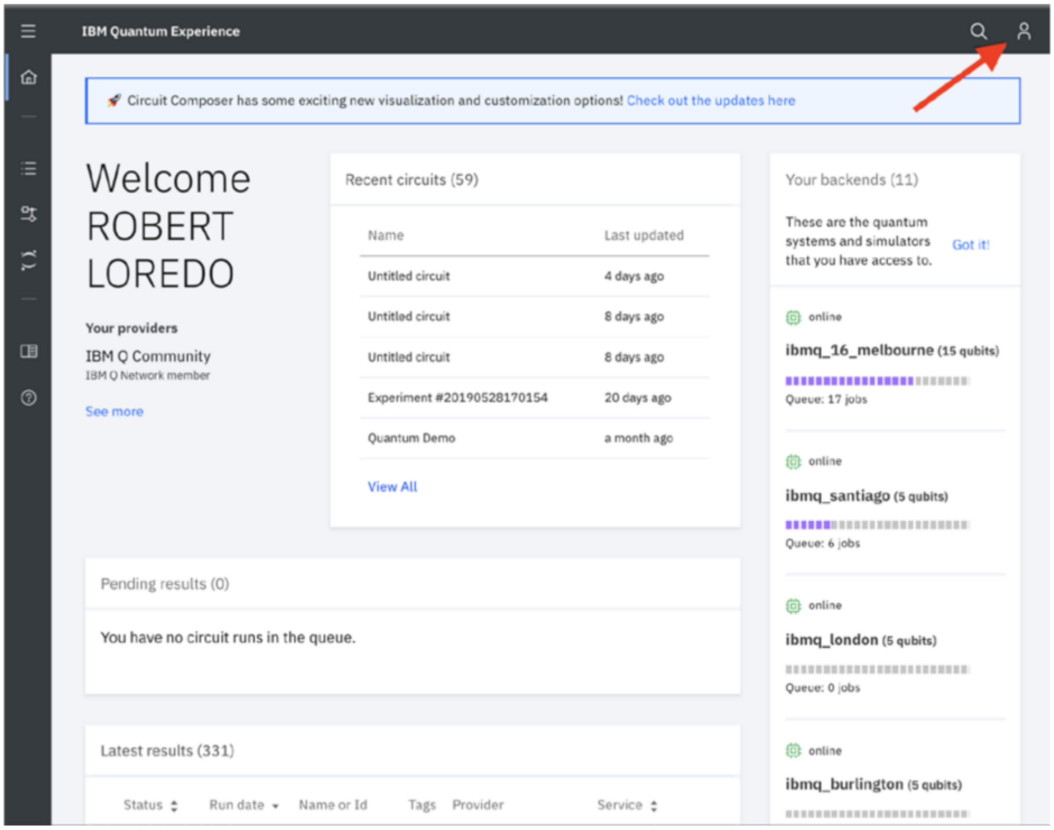


Figure 1.1 – The IBM Quantum Experience sign-in page

3. Once you select the login method of your choice, you will see the login screen for that method. Simply fill out the information, if it's not already there, and select login.

4. Once signed in, you will land on the **Home** page. This is the first page you will see each time you log in to the IBM Quantum Experience site:



Circuit Composer has some exciting new visualization and customization options! Check out the updates here

Welcome ROBERT LOREDO

Your providers

IBM Q Community

IBM Q Network member

See more

Recent circuits (59)

Name	Last updated
Untitled circuit	4 days ago
Untitled circuit	8 days ago
Untitled circuit	8 days ago
Experiment #20190528170154	20 days ago
Quantum Demo	a month ago

View All

Your backends (11)

These are the quantum systems and simulators that you have access to. Got it!

online
ibmq_16_melbourne (15 qubits)
Queue: 17 jobs

online
ibmq_santiago (5 qubits)
Queue: 6 jobs

online
ibmq_london (5 qubits)
Queue: 0 jobs

online
ibmq_burlington (5 qubits)

Pending results (0)

You have no circuit runs in the queue.

Latest results (331)

Status	Run date	Name or Id	Tags	Provider	Service

Figure 1.2 – The IBM Quantum Experience home page

Now that you have registered to the IBM Quantum Experience, let's take a quick tour and delve into some features that make up the IQX home page. Let's start by reviewing the home page, specifically the **Personal** profile tab. You can access your personal profile via your avatar, located at the top right of the page (as pointed out in *Figure 1.2*).

Understanding the Personal profile tab

This section explains the profile of the logged-in user. This is helpful if you have multiple accounts and you wish to keep track of them. The provider limits the number of jobs that can be executed or queued on a given device at any one time to a maximum, as specified in the documentation. There are many ways to access all the various quantum devices; those listed in the open group will see all freely available quantum devices, as illustrated along the right side of *Figure 1.2*. For those who are members of the **IBM Q Network**, you will have access to the open devices, as well as premium quantum devices such as the 65 qubit quantum computer.

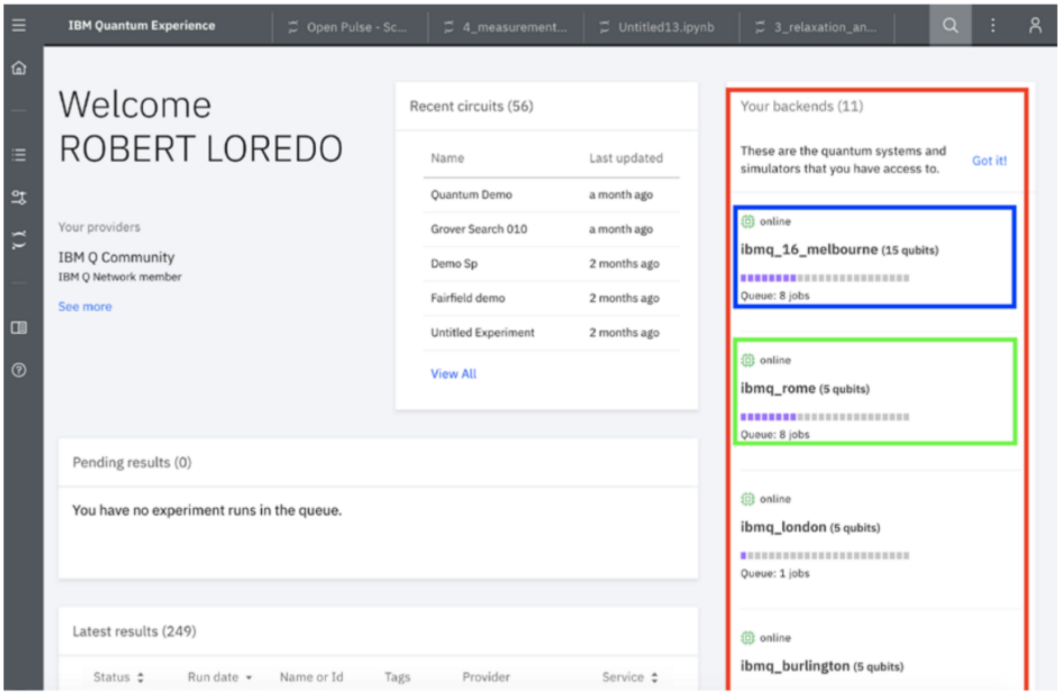
Now that you have completed the sign-up process and successfully logged in, we can start off by taking a tour of the IBM Quantum Experience application. This will be where most of the work within this book will take place, so it will benefit you in understanding where everything is so that you can easily make your way around it while developing your quantum programs.

Getting started with IBM Quantum Experience

This section provides a quick way to launch either **Circuit Composer** or the notebooks located in the Quantum Lab views, herein simply referred to as **Qiskit notebooks**, each of which we will cover in detail in *Chapter 2*, *Circuit Composer – Creating a Quantum Circuit*, and *Chapter 3*, *Creating Quantum Circuits Using Qiskit Notebooks*, respectively, so hang in there. But as with other views, know that you can kick-start either from the main dashboard view or from the left panel. Each button easily provides a quick launch for either of the two circuit generators.

Learning about your backends

This section lists the available backend quantum systems that are provisioned for your use (as shown in *Figure 1.3*). It not only provides a list of the available backends but also provides details for each, such as the *status* of each backend. The status includes whether the device is online or in maintenance mode, how many **qubits** (**quantum bits**) each device contains, and how many experiments are in the queue to be run on the device. It also contains a color bar graph to indicate queue wait times, as illustrated between **ibmq_16_melbourne** and **ibmq_rome** in the following screenshot. Be aware that the quantum devices listed for you may be different from those listed here:



Welcome
ROBERT LOREDO

Your providers
IBM Q Community
IBM Q Network member
See more

Recent circuits (56)

Name	Last updated
Quantum Demo	a month ago
Grover Search 010	a month ago
Demo Sp	2 months ago
Fairfield demo	2 months ago
Untitled Experiment	2 months ago

View All

Pending results (0)

You have no experiment runs in the queue.

Latest results (249)

Status	Run date	Name or Id	Tags	Provider	Service

Your backends (11)

These are the quantum systems and simulators that you have access to. Got it!

online
ibmq_16_melbourne (15 qubits)
Queue: 8 jobs

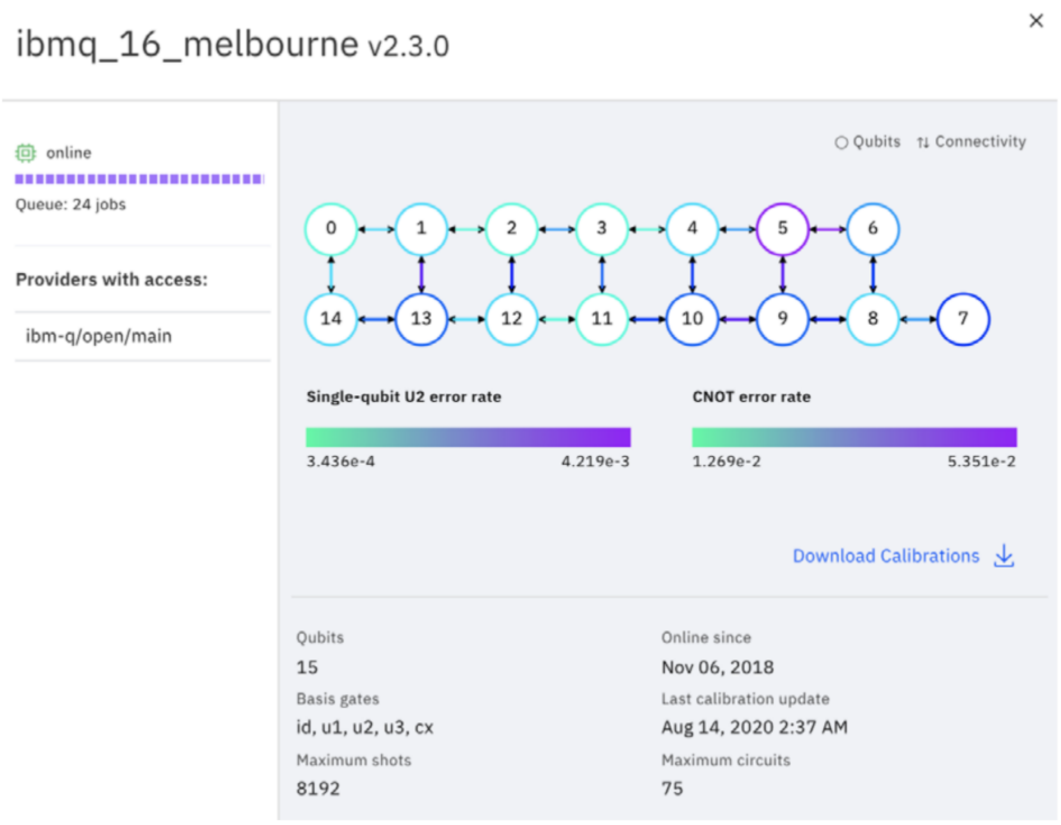
online
ibmq_rome (5 qubits)
Queue: 8 jobs

online
ibmq_london (5 qubits)
Queue: 1 jobs

online
ibmq_burlington (5 qubits)

Figure 1.3 – Provisioned backend simulators and devices

From the preceding screenshot, you can see that another great feature that IQX has with respect to the backend service is the ability to see the hardware details of each real quantum device. If you hover your mouse over each device listed, you will see an expansion icon appear at the top right of the device information block. If you select a device (for example, **ibmq_16_melbourne**), you will see the device details view appear, as shown in the following screenshot:


ibmq_16_melbourne v2.3.0

online

Queue: 24 jobs

Providers with access:

ibm-q/open/main

Qubits
15

Basis gates
id, u1, u2, u3, cx

Maximum shots
8192

Online since
Nov 06, 2018

Last calibration update
Aug 14, 2020 2:37 AM

Maximum circuits
75

Figure 1.4 – Device details view: The status (left) and configuration and error rates (right)

From the previous screenshot, you can see that the device details view contains some very relevant information, particularly if you are working on any experiments that have intricate connectivity between qubits or analyzing error mitigation techniques. On the left of the screenshot (*Figure 1.4*), you can see the basic status information of the device. This is similar to what you see before expanding the device information. In the square on the right, we get into a little more detail with respect to the devices' configuration, connectivity, and error rates.

As described in the shaded bar area, where the error rate range is illustrated by **Single-qubit U3 error rate**, and **CNOT error rate** (single qubit and multi-qubit, respectively), qubits are identified as the circles where the number specifies the qubit number in the device. The arrows in between identify how each qubit is connected to the other qubits. The connections are specific to how the multi-qubit operations are specified.

For example, in the **15** qubit configuration in *Figure 1.4* (on the right), you can see that qubit number **4** is the source for target qubits **3** and **10** (we will get into what source and target mean later, but for now just assume that actions to the target qubit are triggered by the source qubit). You can also see that qubit **4** is the target qubit of qubit **5**. This visual representation is based on information provided by the device configuration, which you can also access programmatically using **Qiskit**.

Another piece of information you can get here is the error rates. The devices are calibrated at least once a day or so, and each time they are calibrated, they calculate the average error rates for a single gate (**u3**) and multi-gates (**CNOT**). The error rates vary per qubit, or qubits for multi-gates, and therefore, the diagram uses a color heat map to identify where the qubit sits on the error rate scale. Each qubit has a different color associated with it. This color makes it possible to visually identify where on the error rate scale that qubit falls. If you are running an experiment on a qubit that requires low error rates, then you can see from this diagram which of these qubits has the lowest error rate when last calibrated.

Below the qubit configuration, you will see a link that also allows you to download the entire configuration information in a spreadsheet. The details there are very specific to each qubit and they provide more information that isn't visible on the qubit configuration diagram.

Finally, at the bottom of the view are the specifics of the device itself, which includes the number of qubits, the date the device went online, and the basis gates available on the device.

You can now close the device configuration diagram to return to the dashboard, where we will next learn about the quantum programs and how to monitor them.

Learning about pending and latest results

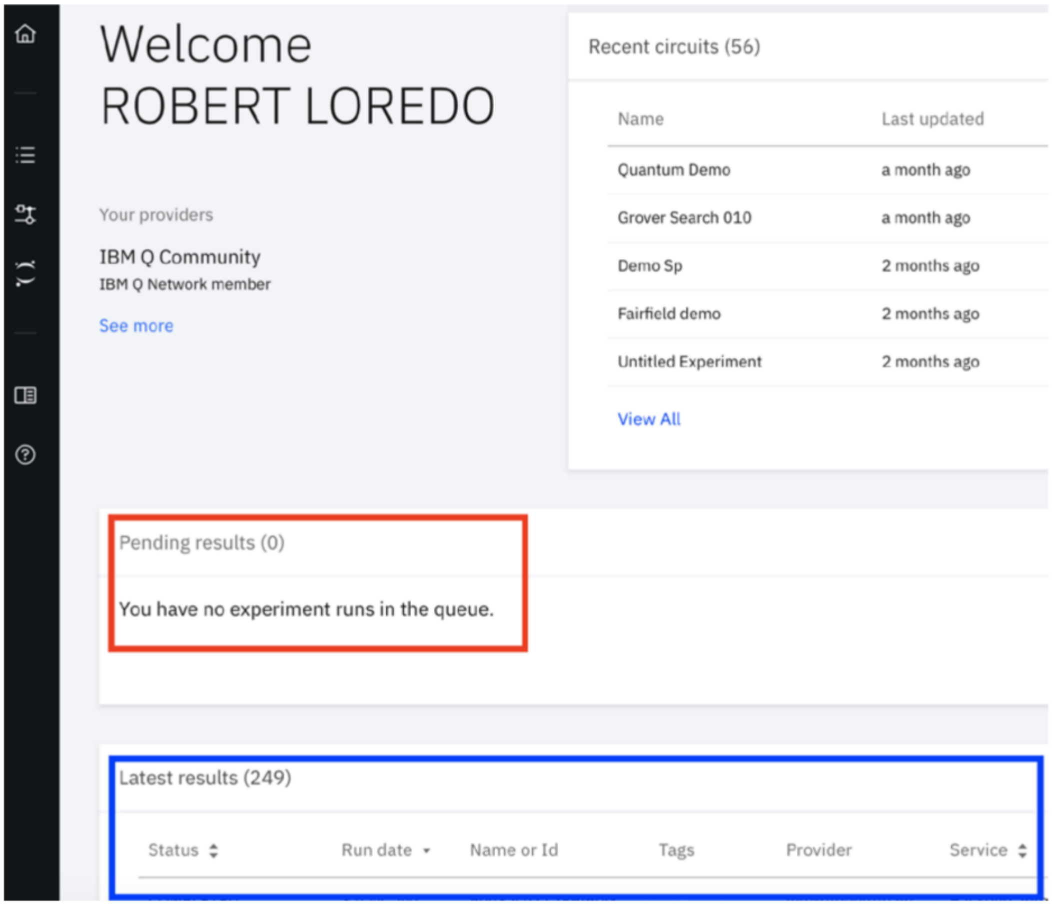
The table shown in *Figure 1.5* contains the experiments that are pending completion on the backend devices. You can use this view to quickly see whether your experiments have run, and if not, where in the queue your experiment is set to run next.

Under your pending results table is the table where all your latest results are stored. These are the last few experiment results that were run on either the simulator or real devices on the backend. Each device is initially sorted by creation date but can be sorted by either backend or status, if need be.

Important Note

Details regarding job objects will be covered in *Chapter 9, Executing Circuits Using Qiskit Aer*.

As well as this, the job ID is listed so that you can call back the details from that job at a later time, as seen in the following screenshot:



Welcome
ROBERT LOREDO

Your providers

IBM Q Community
IBM Q Network member

See more

Recent circuits (56)

Name	Last updated
Quantum Demo	a month ago
Grover Search 010	a month ago
Demo Sp	2 months ago
Fairfield demo	2 months ago
Untitled Experiment	2 months ago

View All

Pending results (0)

You have no experiment runs in the queue.

Latest results (249)

Status	Run date	Name or Id	Tags	Provider	Service

Figure 1.5 – Pending results and latest results

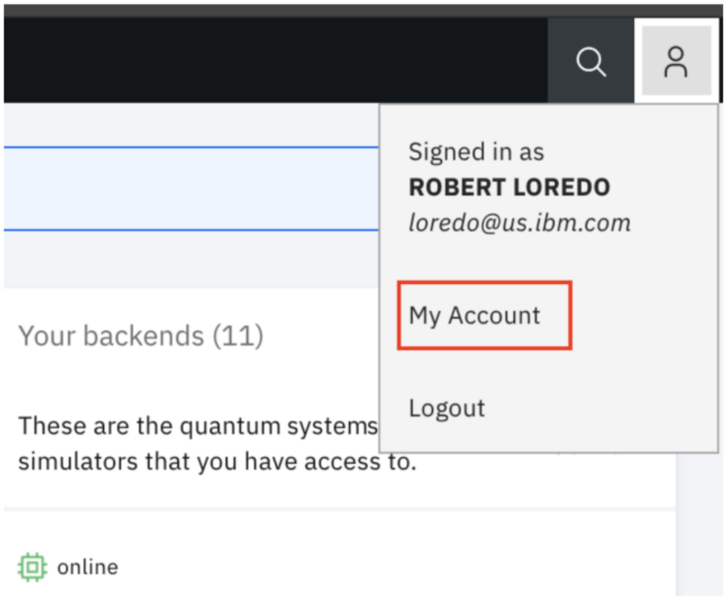
In this section, you have learned where to find information about your experiments, hardware details about the simulators and the real quantum devices. Next, we will explore your account profile.

Exploring My Account

In this section, you will explore your account details view, where you will find information about your account and what services are available to you. This includes services such as the ability to view the list of backend systems available to you, notification settings, and resetting your password.

To open the account view, follow these steps:

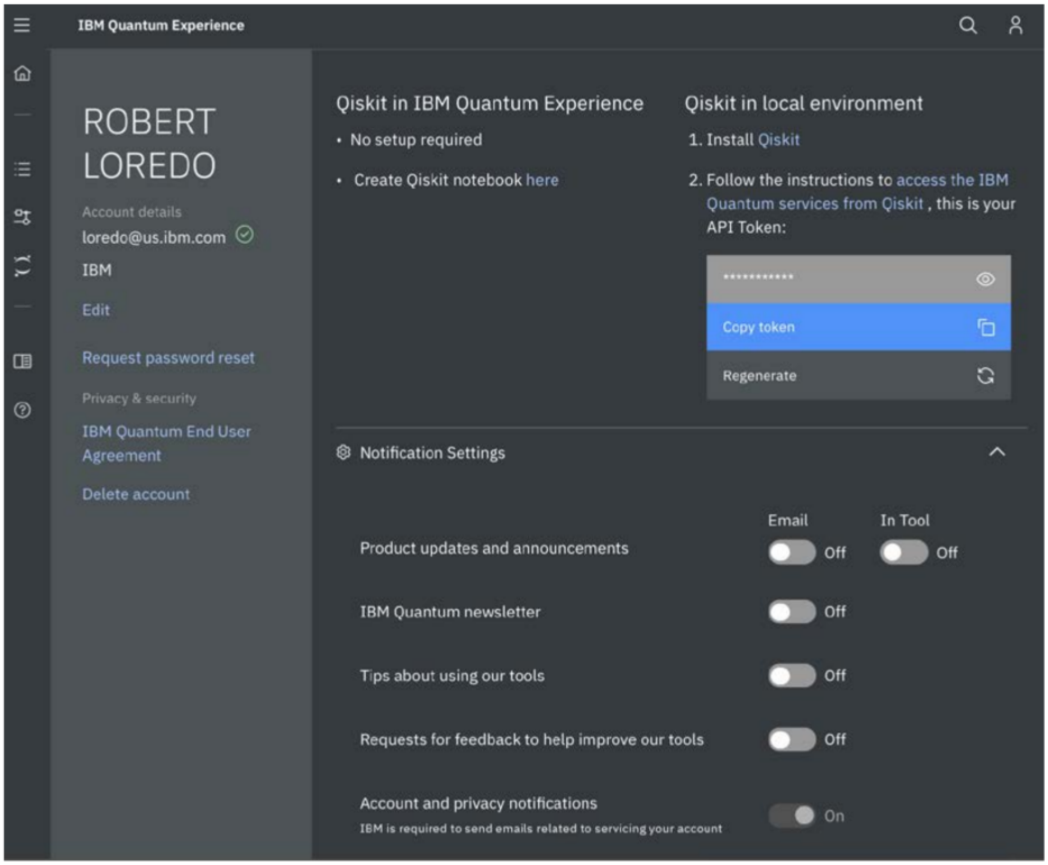
1. Click on your avatar at the top right of the dashboard (as highlighted in the following screenshot) and select **My Account**:



These are the quantum systems
simulators that you have access to.

Figure 1.6 – The My Account option on the dashboard

2. Once the **My Account** view is loaded, you will see a page similar to this:


IBM Quantum Experience

ROBERT LOREDO

Account details

loredo@us.ibm.com

IBM

Edit

Request password reset

Privacy & security

IBM Quantum End User Agreement

Delete account

Qiskit in IBM Quantum Experience

- No setup required
- Create Qiskit notebook here

Qiskit in local environment

1. Install Qiskit
2. Follow the instructions to access the IBM Quantum services from Qiskit , this is your API Token:

****

Copy token

Regenerate

Notification Settings

	Email	In Tool
Product updates and announcements	Off	Off
IBM Quantum newsletter	Off	
Tips about using our tools	Off	
Requests for feedback to help improve our tools	Off	
Account and privacy notifications IBM is required to send emails related to servicing your account	On	

Figure 1.7 – The My Account view

From the preceding screenshot, you can see that on your account page, you will see the following information sections:

- **Account details**: This section has your account and contact information that you used to register. It also includes options such as resetting your password, privacy and security information, and the option to delete your account.
- **Qiskit in IBM Quantum Experience**: This includes a quick link to launch a Qiskit notebook to run your experiments. We will review the Qiskit notebook later in this book, but for now, just know that you can launch a Qiskit notebook from here as well.

- **Qiskit in local environment**: This section allows you to install Qiskit and run experiments from your local machine without the need to connect to IQX via the cloud. This is exceptionally helpful when you wish to run experiments but do not have access to a network. By running experiments from your local machine, this allows you to run simulators that are installed as part of the Qiskit installation. However, keep in mind that in order to run the experiments on a real quantum device, you will need network connectivity to those real devices.

  If you want to run the experiments on a real device from your local machine, then you will need to copy the token (highlighted in *Figure 1.7*) that was generated for you in the background. You should then assign it to the **Qiskit IBMQ provider** class. Details of the IBMQ provider class will be discussed in *Chapter 9*, *Executing Circuits Using Qiskit Aer*, but for now, this is where you can copy the **Application Programming Interface** (**API**) token.

  Also, note that there is an option to regenerate the API token. If you choose to regenerate the token, you will need to delete your old token and save the regenerated one in your local IBMQ provider class. The save account method of the IBMQ provider class will persist the value in your local machine, so you will only have to save it once and then load the account each time you wish to use a real quantum device for your experiment.

  Since this book is written primarily for use on the IBM Quantum Experience site, we will cover running and setting up on your local machine. Just in case you happen to not have network connectivity, you can still run simulated experiments locally.

- **Notification Settings**: This section simply allows you to set your notifications and how you prefer to receive information, such as when experiments have completed or other information or surveys that you wish to contribute.
- **Your accounts**: This last section toward the bottom of the **My Account** view is an overview of the accounts that you have and a list of the provisioned systems you have access to. These provisions are selected and assigned as part of the sign-up process. This includes information such as when you first signed up, the project that you are associated with (**main** is usually the default project), provider information, and the allocated backend systems that you have access to. These allocated backends that you can see are either real devices, such as **ibmq_16_melbourne**, or simulators, such as **ibmq_qasm_simulator**, which are running on the IQX cloud. We will discuss the details of the simulators and devices in later chapters.

Now that we are done with our tour of the IBM Quantum Experience layout, we're ready to get to work. In the following chapters, we will delve into each section and progress to writing quantum programs.

Summary

In this chapter, we reviewed the dashboard, which provides plenty of information to help you get a good lay of the land. You now know where to find information regarding your profile, details for each of the devices you have available, the status of each device, as well as the status and results of your experiments.

Knowing where to find this information will help you monitor your experiments and enable you to understand the state of your experiments by reviewing your backend services, monitoring queue times, and viewing your results queues.

You also have the skills to create an experiment using either Circuit Composer or the Qiskit notebooks. In the next chapter, we will learn about Circuit Composer in detail.

Questions

1. Which view contains your API token?
2. Which device in your list has the fewest qubits?
3. How many connections are there in the device with the fewest qubits?
4. What are the two tools called that are used to generate quantum circuits?
5. Which view would provide you with the list of basis gates for a selected device?

2
Circuit Composer – Creating a Quantum Circuit

In this chapter, you will learn how to use the **Circuit Composer** and what each of the composer's component functions are with respect to creating and running experiments. The composer will help you to visually create a quantum circuit via its built-in user interface, which in turn will help you to conceptualize how the basic principles of quantum mechanics are used to optimize your experiments. You will also learn how to import quantum circuits, preview the results of each experiment, and create your first quantum circuit.

The following topics will be covered in this chapter:

- Creating a quantum circuit using the Composer
- Creating our first quantum circuit
- Building a coin-flipping experiment

By the end of this chapter, you will know how to create a quantum circuit using the **Graphical Editor** to create experiments that simulate classic gates and some quantum gates. You will also learn where to examine the various results of your experiment, such as state vectors and their probabilities. This will help you understand how some quantum gate operations affect each qubit.

Technical requirements

In this chapter, some basic knowledge of computing is assumed, such as understanding the basic gates of a classic computing system; for example, **bit flip** (0 to 1), **NOT gates**, and so on. Here is the full source code used throughout the book: `https://github.com/PacktPublishing/Learn-Quantum-Computing-with-Python-and-IBM-Quantum-Experience`. Here is the link for the CiA videos: `https://bit.ly/35o5M80`

Creating a quantum circuit using the Composer

In this section, we will review the Composer layout so that you can understand the functionality and behavior of the Composer when creating or editing your quantum circuits. Here, you will also create a few circuits, leveraging the visualization features from the Composer to make it easy for you to understand how quantum circuits are created. So, let's start at the beginning: by launching the Composer editor.

Launching the Composer editor

To create a quantum circuit, let's first start by opening up the Circuit Composer. To open the Composer view, click on the Circuit Composer icon located on the left panel as shown in the following screenshot:

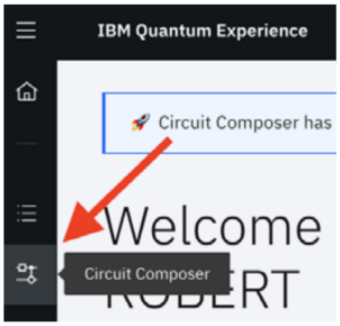


Figure 2.1 – Launching the Circuit Composer (left panel)

Now that you have the Composer open, let's take a tour of what each component of the Composer editor provides you with.

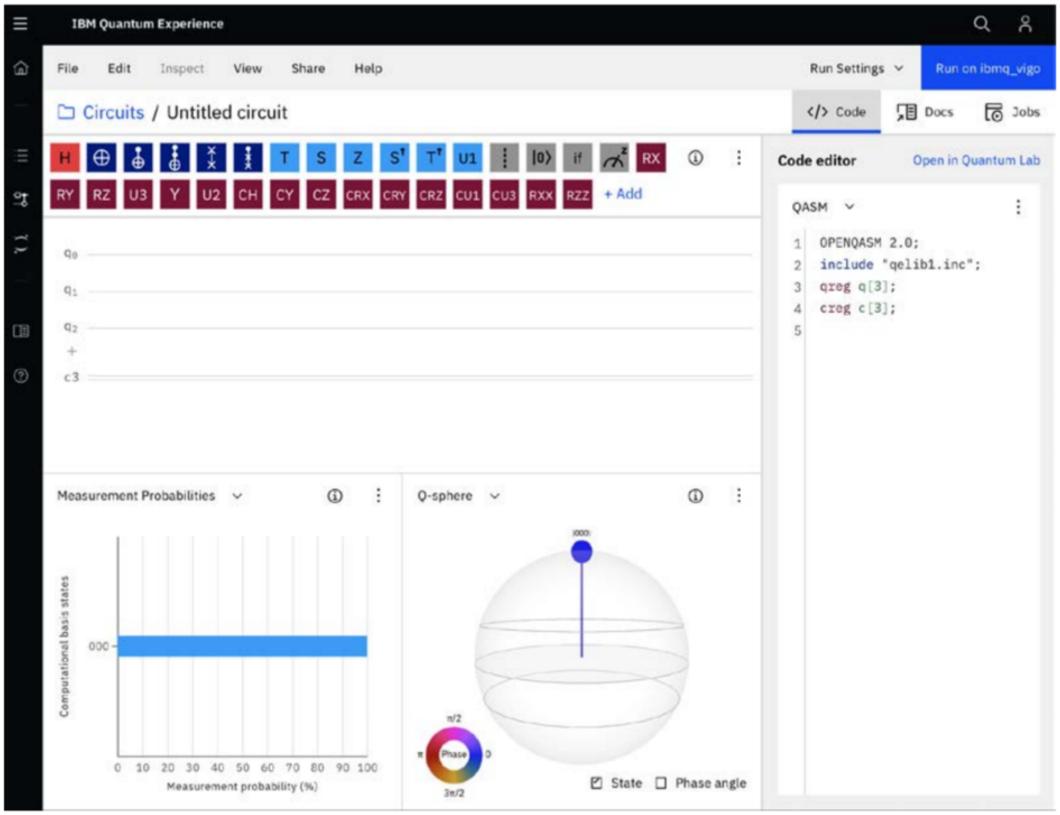
Familiarizing yourself with the Circuit Composer components

In this section, we will get familiar with each of the components that make up the Composer. Each of these has features specific to the various components of the Composer editor. These can provide insights by allowing you to do things such as visually inspecting the results of your experiments by displaying the results in a variety of ways. Visualizing the construction of the quantum circuit will help you conceptualize how each quantum gate affects a qubit.

Understanding the Circuit Composer

In this section, we will review the various functionalities available to ensure you have a good understanding of all the different features available to you.

In the following screenshot, you can see the landing page of the **Circuit Composer** editor view:



```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c[3];
```

Figure 2.2 – Circuit Composer view

From the preceding screenshot, you can see at the center of the page is the **Circuit Composer** view. In the following screenshot, you can see a series of **gates** and **operations**:

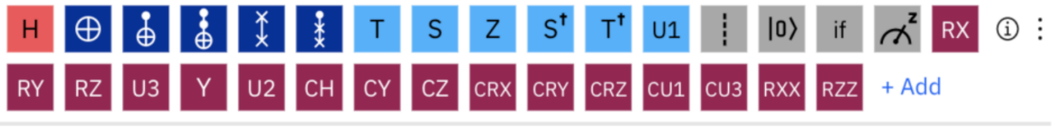


Figure 2.3 – Gates and Operations

As you can see in the preceding screenshot, each of these components has a specific function or operation that acts upon the qubit(s), which we will cover in detail in *Chapter 6, Understanding Quantum Logic Gates*.

As we can see in the following screenshot, below the collection of gates and operations, we have the Circuit Composer itself:

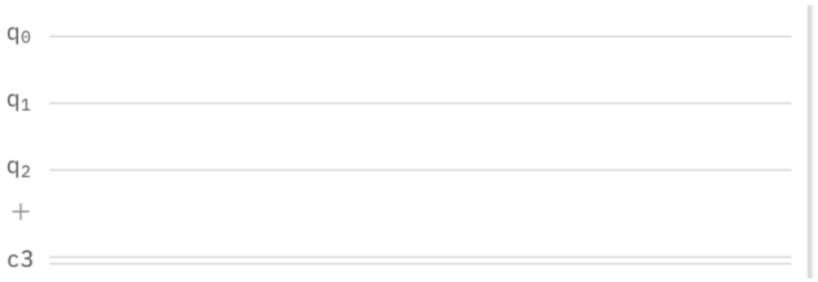


Figure 2.4 – Circuit Composer

As you can see from the preceding screenshot, the default circuit includes three qubits, each of which is labeled with a **q**, and the index appended in order from least significant bit (in this case, **q**$_2$, **q**$_1$, **q**$_0$). Each qubit is initialized to an initial state of **0** before running the experiment.

Next to the qubit you will see a line, which looks like a wire running out from each qubit, in the circuit:

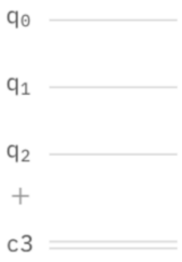


Figure 2.5 – Qubits and circuit wires

As you can from the preceding screenshot, this line is where you will be creating a circuit by placing various gates, operations, and barriers on them. The circuit has three wires, each of which pertains to one of the three qubits on the quantum computer. The reason it is called a Composer is primarily due to the fact that it looks very similar to a music staff used by musicians to compose their music. In our case, the notes on the music staff are represented by the gates and operations used to ultimately create a quantum algorithm.

In the next section, we will review the various options you have available to customize the views of the IQX. This will allow you to ensure you can only see what you need to see while creating your quantum program.

Learning how to customize your views

Continuing with our Composer tour, at the top of the Composer view is the circuit view menu option that allows you to save your circuit, clear the circuit, or share your quantum circuit. First, we will cover how to save your circuit. To do this, simply click on the default text at the top left of the circuit composer where it currently reads **(Untitled circuit)** and type in any title you wish. Ideally, select a name that is associated with the experiment. In this case, let's call it **My First Circuit** and save it by either hitting the *Enter* key or clicking the checkmark icon to the right of the title.

Across the top of the Composer, you will see a list of drop-down menu options. We can see these in the following screenshot:

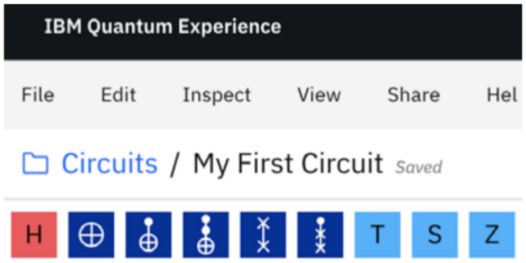


Figure 2.6 – Composer menu options

The menu items in the preceding screenshot have the following options:

- **File** provides options to create, copy, open a new circuit, or view all quantum circuits.
- **Edit** allows you to manage your circuit, clear all operators, and edit the circuit description.
- **Inspect** provides the ability to step through your circuit, similar to debug mode.

- **View** enables the various view options.
- **Share** allows you to share your quantum circuit with others.
- **Help** provides various guides, tours, and content related to quantum computing.

Let's now take a look at each of the various views in the following sections.

The Graphical Editor view

The **Graphical Editor** view contains a few components used to create quantum circuits. Which includes the following:

- **Circuit Composer**: UI components used to create quantum circuits
- **Gates and Operators**: A list of available drag and drop gates and operators available to generate a quantum circuit
- **Options**: A list of options such as the gate glossary, collapse gates, and options for downloading an image representation of your quantum circuit

The following is a screenshot illustrating each of the preceding components:

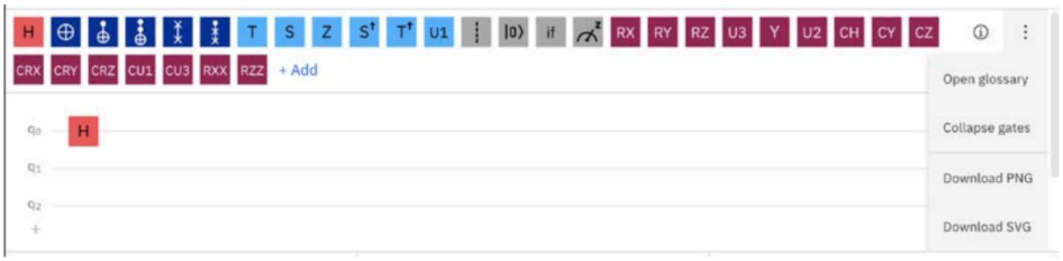


Figure 2.7 – Graphical Editor view

Now that we know where we can create a quantum circuit, let's move on to displays, which provide the results of our quantum circuit.

The Statevector view

The **Statevector** view allows you to preview the state vector results of your quantum circuit. The state vector view presents the computational basis states along the *x* axis, and the **Amplitude** along the *y* axis. In this case, since we do not have any gates or operators on our circuit, the state vector representation is that of the initial state. Where the initial state indicates that all qubits are initialized to the 0 (zero) state and with an amplitude of 1, we see that presented in the following screenshot:

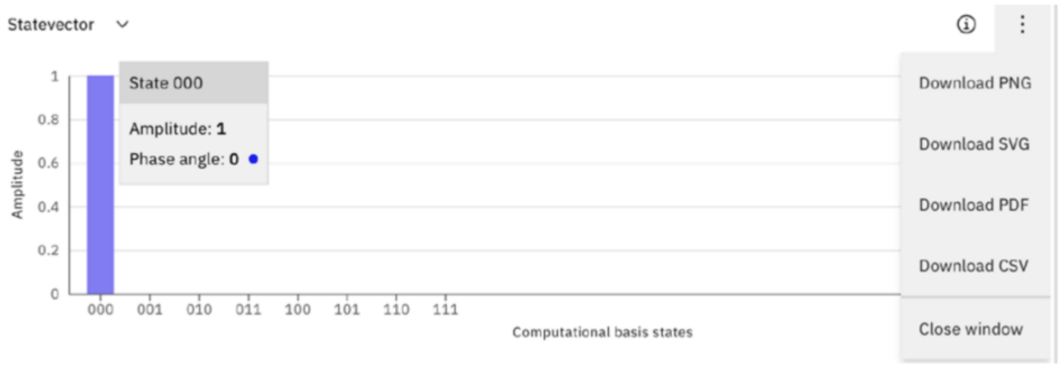


Figure 2.8 – Statevector view

Other options available to us include various ways to download the state vector information, as illustrated in the drop-down menu at the top right of the previous screenshot.

The state vector information is just one of the visual representations of your quantum circuit. There are a couple of others we want to visit before moving on.

The Measurement Probabilities view

The next view is the **Measurement Probabilities** view. This view presents the expected measurement probability result of the quantum circuit. As mentioned in the previous description, and illustrated in the following screenshot, since we do not have any operators on the circuit, the results shown are all in the initial state of 0:

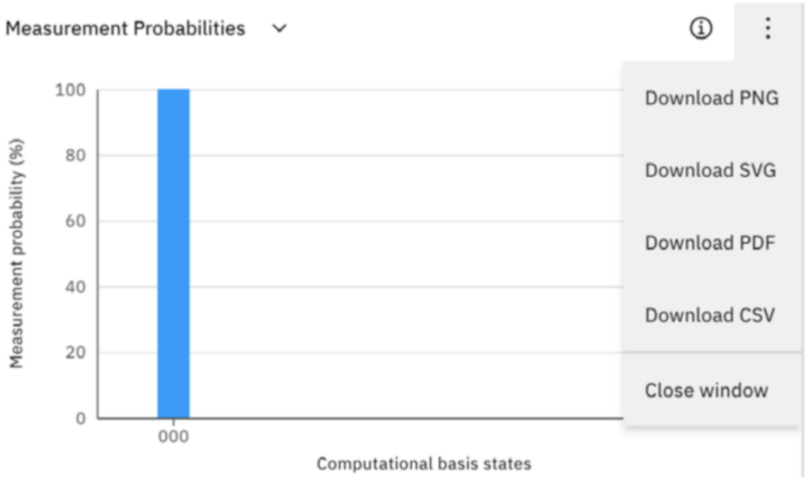

Measurement Probabilities
Measurement probability (%)
100
80
60
40
20
0
000
Computational basis states
Download PNG
Download SVG
Download PDF
Download CSV
Close window

Figure 2.9 – Measurement Probabilities view

The options here also provide various formats to download the measurement probabilities.

The Q Sphere view

Finally, the last of the state visualizations we have to review is the **Q Sphere** view. The Q sphere is similar to the Bloch sphere; however, the Bloch sphere does have some limitations, particularly when working with more than one qubit. The Bloch sphere is used to represent the vector of the current state of a qubit. The Q sphere can be used to represent the state information of a single qubit or multiple qubits, including the phase information. The following screenshot shows a representation of the three qubits we have in our circuit, all of which are in the initial state:

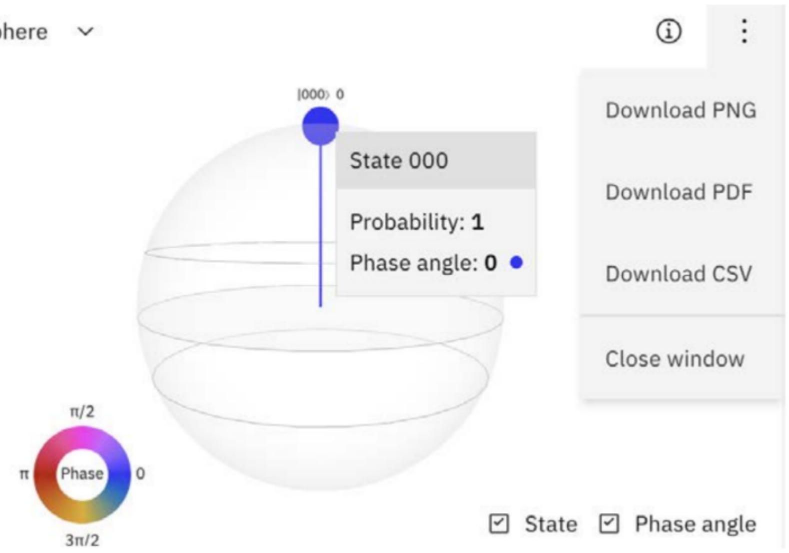


Figure 2.10 – Q Sphere view

The Q-sphere view has two components, the first is the Q-sphere itself that captures the state vector of the various qubit states represented by a vector that originates at the center of the sphere. At the end of the vector is a smaller sphere, which represents the details of the state. The states represented by these small spheres are visible when hovered over. The previous screenshot illustrates the 3 qubits in an initial state of 000, with a probability of 1, and a phase angle of 0.

The second component is located at the bottom left, which is the legend that describes the phase of the states. Since the small sphere represents the phase angle of 0, the color of the sphere is blue, which is the same that the legend indicates for the phase of 0. If the states were out of phase by a value of π, then the color of the sphere would be red.

There are various options here; to the top right you have various options to download visualizations in different image formats, and at the bottom right you can select whether to enable the state or phase angle information of the Q-sphere.

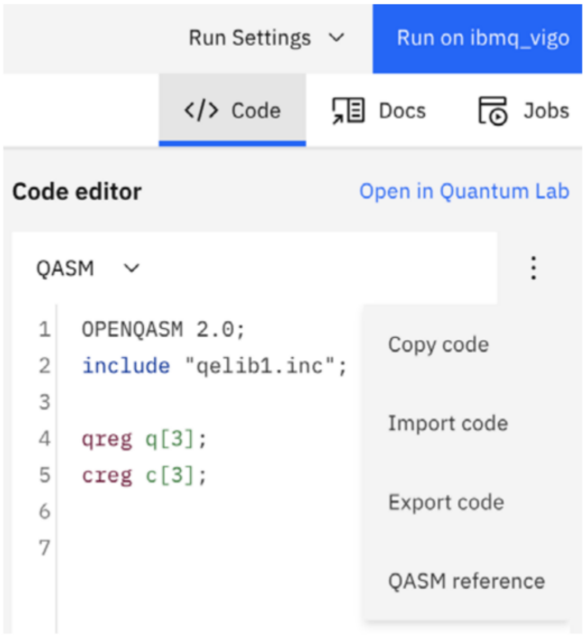
One last thing to note is at the top left, you can see a dropdown that allows you to switch between all the views we reviewed, such as the measurement probabilities and state vector.

Now that we are familiar with the various state representation views, let's look at the last view that allows us to write code and execute our quantum circuits.

The Code Editor view

The last view we will cover here is the **Code Editor** view. Here we can write code to build the circuit itself. At the top of the Code Editor view there are three tabs, namely, **Code**, **Docs**, and **Jobs**. Each tab displays details about itself.

The **Code** tab has the code editor itself, which you can use to code using **QASM** or Qiskit code, for which you make your selection with the drop-down menu at the top left of the editor. The options available in the Code Editor provide a way to copy, import, and export code. Also included is the QASM reference link, which redirects you to details of the QASM language. The following screenshot illustrates the **Code editor** view with the options expanded:



```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];
```

Figure 2.11 – Code Editor view

The **Docs** tab displays the documentation available and the **Jobs** tab displays your pending and completed job running on the simulators or quantum devices.

In this section, we learned about how to create a quantum circuit using the Composer. We also learned about the views and components of the Circuit Composer views.

Now that you have an understanding of the various views and components that make up the Circuit Composer views, we can start creating our first quantum circuit and leveraging a lot of these views.

Creating our first quantum circuit

Now that we know where everything is in the Circuit Composer, we will create our first quantum circuit. This will help you to get a better understanding of how all these components work together and it will show you how these components provide insights such as current state and probabilistic estimation as you build your first quantum experiment.

Building a quantum circuit with classical bit behaviors

We are all familiar with some of the basic classic bit gates such as **NOT**, **AND**, **OR**, and **XOR**. The behavior that these classic gates perform on a bit can be reproduced on a quantum circuit using quantum gates. Our first experiment will cover these basic building blocks, which will help you to understand the correlation between quantum and classic algorithms.

Our first experiment will be to simulate a NOT gate. The NOT gate is used to flip the value, in this case from $|0\rangle$ to $|1\rangle$, and vice versa. The gate we will use to do this is the **NOT gate**. We will cover details on how this gate operates on qubits in *Chapter 6*, *Understanding Quantum Logic Gates*.

To simulate the NOT gate on a quantum circuit, follow these steps:

1. From the open composer circuit that you previously created and titled **My First circuit**, click and drag the **NOT** gate, which is visually represented by the ⊕ symbol, from the list of gates down onto the first qubit, as shown in the following screenshot:

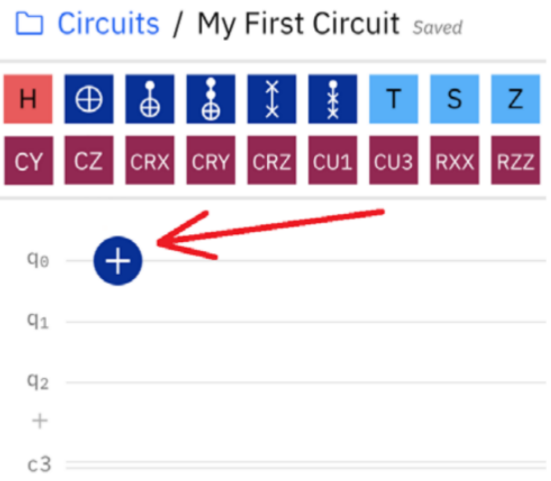


Figure 2.12 – Add an X (NOT) gate to the first qubit

2. Next, click and drag the measurement operation onto the first qubit, just after the **NOT** gate. By taking a measurement of the qubit and having its value sent out to the pertaining classic bit, we are essentially reading the state of the qubit.

   A measurement occurs when you want to observe the state of the qubit. What this means is that we will collapse the state of the qubit to either a 0 or a 1. In this example, it is pretty straightforward that when we measure the qubit after the **NOT** gate, the reading will be 1. This is because since the initial state is set to 0, applying a NOT gate will flip it from 0 to 1. Therefore, we expect the measurement to read 1.

3. Click and drag another measurement operation onto the second qubit. We'll do this just to contrast the difference between what we would see when we measure a qubit in the initial state, and after a **NOT** gate.

4. Before we run this experiment, let's note a few things. First, note that the classic bits are all on one line (as shown in the following screenshot). This is mostly to save space. In lieu of having three additional wires where each represents a classic bit, a single wire is used to denote the classic bits. They are labeled **c3** to indicate a set of three classic bits:

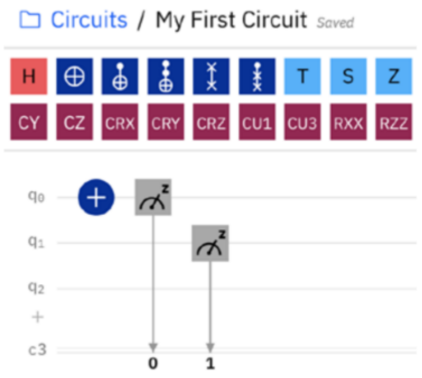
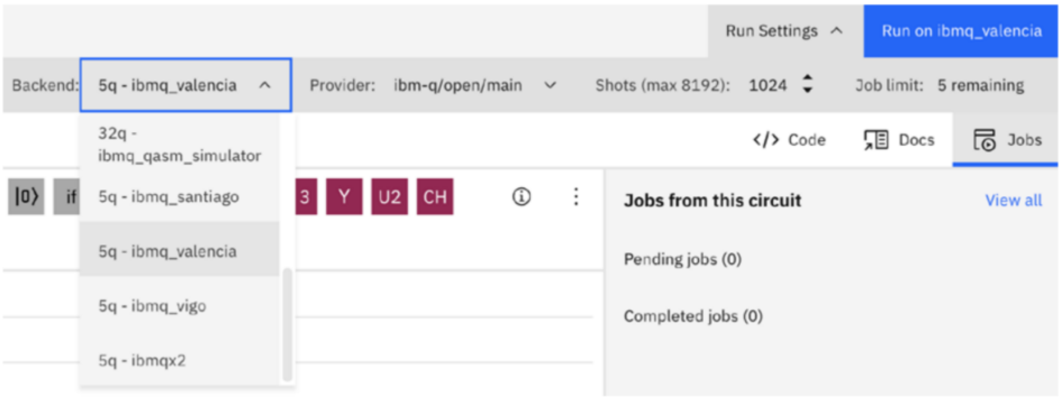


Figure 2.13 – Add a measurement operation to the first qubit

The second thing to notice is that the measurement operations match the qubit number to the classic bit number; in this case, qubit **0** will read out to bit **0**, and qubit **1** will read out to bit **1**, where bit **0** is the least significant bit.

5. Select the **Run Settings** drop-down option located at the top right of the circuit composer view. This will display the run settings, illustrated as follows:



Run Settings | Run on ibmq_valencia

Backend: 5q - ibmq_valencia | Provider: ibm-q/open/main | Shots (max 8192): 1024 | Job limit: 5 remaining

Jobs from this circuit | View all

Pending jobs (0)

Completed jobs (0)

Figure 2.14 – Run settings drop-down view

6. The run dialog provides you with three options:

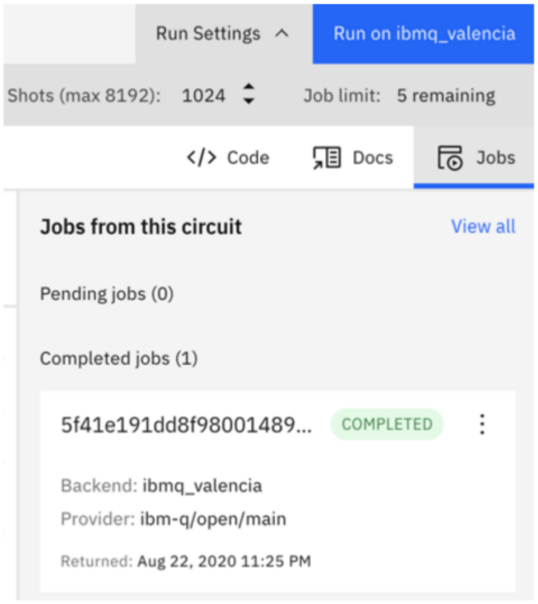
   First, to select which backend device you would like to run the experiment with, the choices are either on a simulator called **ibmq_qasm_simulator** or on an actual quantum device. Select any of the options you wish to run. In this example, we'll select **ibmq_valencia**.

   The second option allows you to select the **Provider**. There are different providers – the **open/main** is for the open free quantum devices, and if you are a member of the **IBM Q Network** then you'll have a provider that assigns you to the available premium quantum devices. For now, leave it at the default setting.

   The last option allows you to select how many shots of the quantum circuit you wish to run. What this means is how many times you wish the quantum circuit to run during your experiment. For now, since this is a simple experiment, let's simply set it to the default value, `1024`.

7. Now that you have selected your run options, let's run the circuit. Click **Run on ibmq_valencia**. If you selected a different device, it will indicate it accordingly.

8. Once your experiment begins, you should see an entry of this experiment in the **Pending Jobs** view to the right of the Composer view. This indicates that your experiment is pending. Once completed, you will see it in the **Results view** shown as follows:



Shots (max 8192): 1024 Job limit: 5 remaining

Jobs from this circuit

Pending jobs (0)

Completed jobs (1)

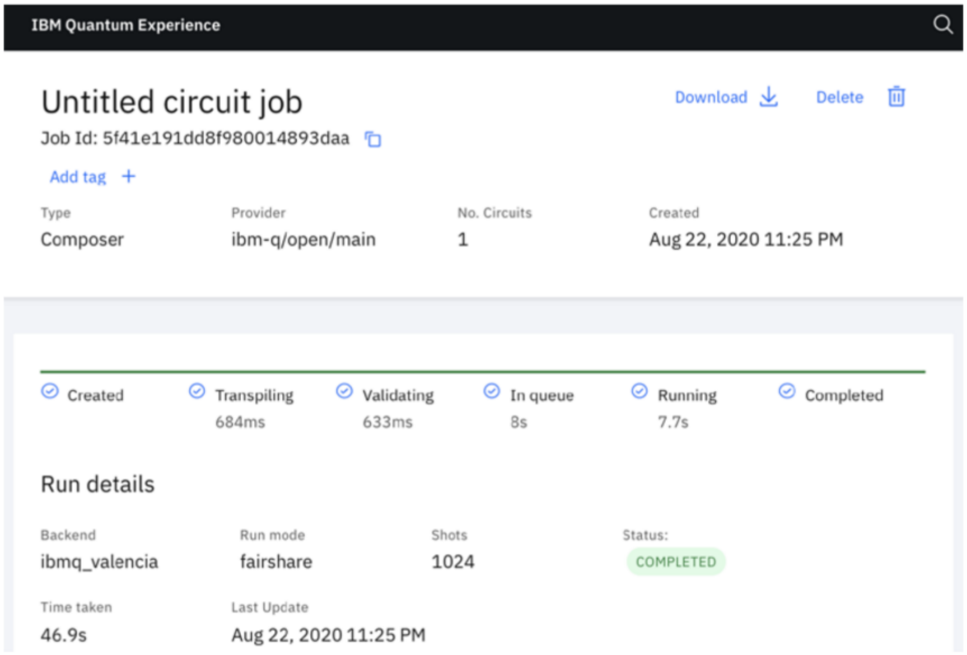
5f41e191dd8f98001489... COMPLETED

Backend: ibmq_valencia
Provider: ibm-q/open/main
Returned: Aug 22, 2020 11:25 PM

Figure 2.15 – Results view displaying pending and completed jobs for the selected circuit

While the job is in the **Pending jobs** list, it will display the status of the job. Once completed, it will automatically move from the **Pending jobs** to the **Completed jobs** list.

9. Upon completion, open your experiment from the **Completed jobs** list by clicking on the job. This opens the experiment results view; you will see details regarding your experiment at the top of the report, as illustrated in the following screenshot:



Untitled circuit job

Job Id: 5f41e191dd8f980014893daa

Type	Provider	No. Circuits	Created
Composer	ibm-q/open/main	1	Aug 22, 2020 11:25 PM

Created	Transpiling	Validating	In queue	Running	Completed
	684ms	633ms	8s	7.7s	

Run details

Backend	Run mode	Shots	Status:
ibmq_valencia	fairshare	1024	COMPLETED

Time taken	Last Update
46.9s	Aug 22, 2020 11:25 PM

Figure 2.16 – Completed job result overview

This view provides details about the results such as the **Time taken** during each task, the **Backend** system it had run on, the number of **Shots**, the current **Status**, and the total time taken to execute. As you look further down the view you will see a histogram of the results from the circuit you just ran on the backend, as illustrated in the following screenshot:

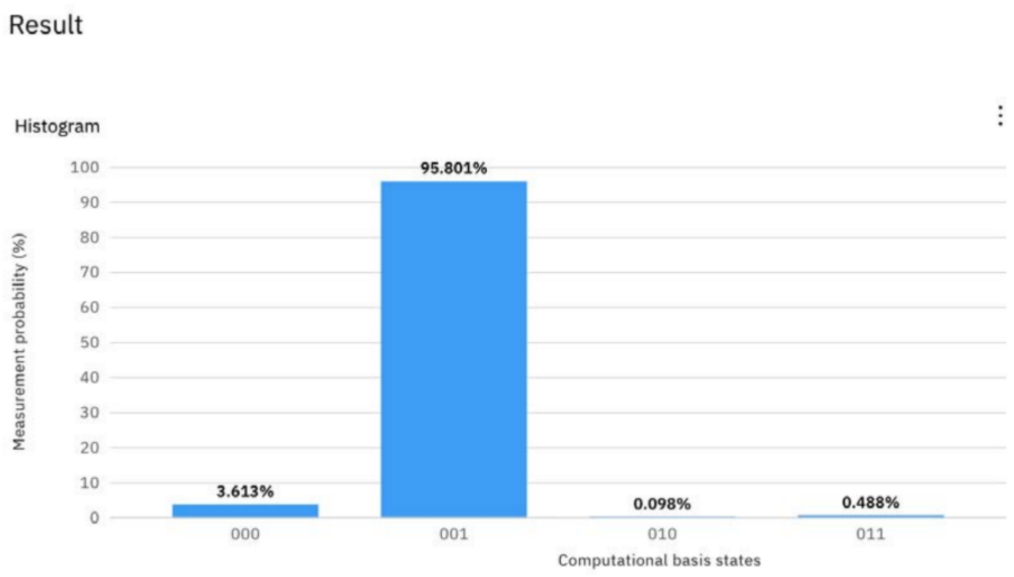


Figure 2.17 – Histogram representation of the circuit results

When you further scroll down the view/page you will see the diagram of the circuit you created, illustrated in the following screenshot:

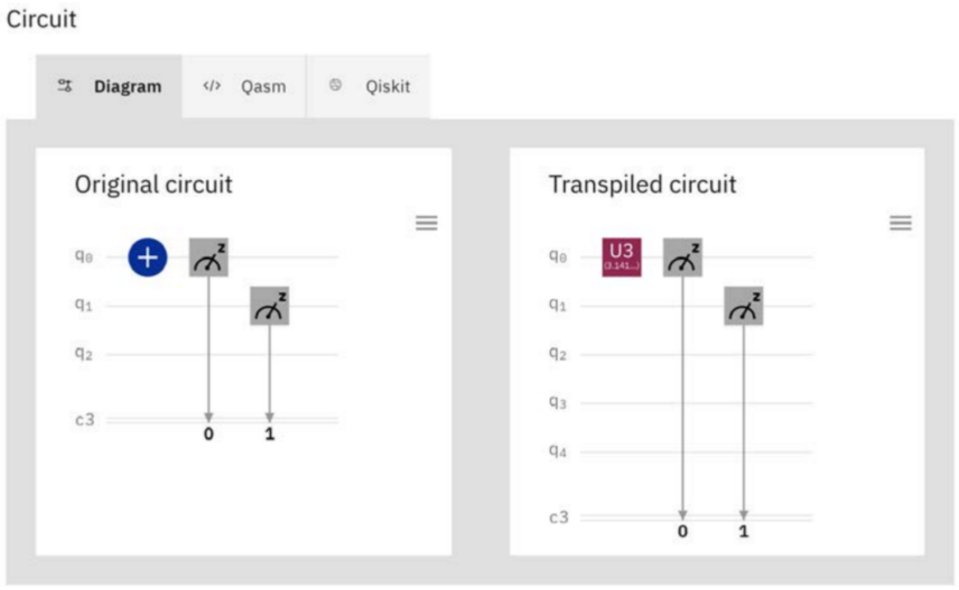


Figure 2.18 – Circuit diagram of the circuit

The diagram of the circuit is just one of the three representations of the circuit. The other two tabs will display the QASM and Qiskit representations.

Now that we have the results from running our first quantum circuit, let's take a closer look at our results and see what we got back.

Reviewing your results

The histogram result in *Figure 2.21* provides information about the outcome of your experiment. Some parts might seem straightforward, but let's review the details.

It may seem trivial now, but later on when we work on more elaborate quantum algorithms, understanding the results will prove invaluable.

There are two axes to the results. Along the *x* axis, we have all the possible states of our circuit. This is what the measurement operations observed when measuring the qubits. Recall that we measured the first and second qubits, so from least significant bit (on the far right), we see that the first two bits are set to 1 and 0 respectively. We know that this is correct due to the fact we placed a NOT gate on the first qubit, which changes its state from 0 to 1. For the second qubit, we simply took a measurement that equates to simply measuring the initial state, which we know to be 0.

The *y* axis provides the probability of the measurement. Since we ran the experiment `1024` times, the results show that we have approximately a 95% probability of the first qubit resulting in the state of **001**. The reason why the probability is 95% and not 100% is due to noise from the quantum device. We will cover the topic of noise in later chapters, but for now we can be confident to a pretty high of probability that the NOT gate worked.

So, when would the probability be different? We'll explore this in the following experiment.

In this section, we simulated a simple NOT gate operation on a qubit and ran the circuit on a quantum device. Pretty simple and straightforward. So now that you were able to create and run your first quantum program, let's start learning something a little more interesting than just changing the state of a qubit.

Building a coin-flipping experiment

If you've ever taken a course in probability and statistics, you might have seen the coin flip example. In this example, you are given an unbiased coin to flip multiple times and track the results of each flip (experiment) as either heads or tails. What this experiment illustrates is that with an unbiased coin and enough samples, you will see that the probability of either heads or tails start to converge to about 50%.

This means that, after running a sufficient number of experiments, the number of times the coin lands on heads becomes very closely equal to the number of times that it lands on tails.

Let's take a moment to make an important note regarding the previously stated analogy with respect to the reality of the preceding experiment. It has been proven that in many ways, any coin could be easily made biased so that when it is flipped, it can land on the same side each time.

That being said, I want to ensure that this is a basic example of an attempt to create a classical analogy of a quantum computing principle in order to get an understanding of the experiment we will be creating, and not to insinuate that this classical experiment equates to a quantum experiment. I will cover these differentiations as we create the next circuit that will simulate flipping a coin over 1,000 times. Let's give this a try:

1. Open the Composer Editor and create a new blank circuit.
2. Click and drag the Hadamard (H) gate onto the first qubit.
3. Click and drag the measurement operation onto the first qubit after the H gate. This will indicate that you wish the value of this qubit to be measured, and assign its resulting value of either 1 or 0 to the corresponding classic bit; in this case, the bit at position 0, as shown in the following screenshot:

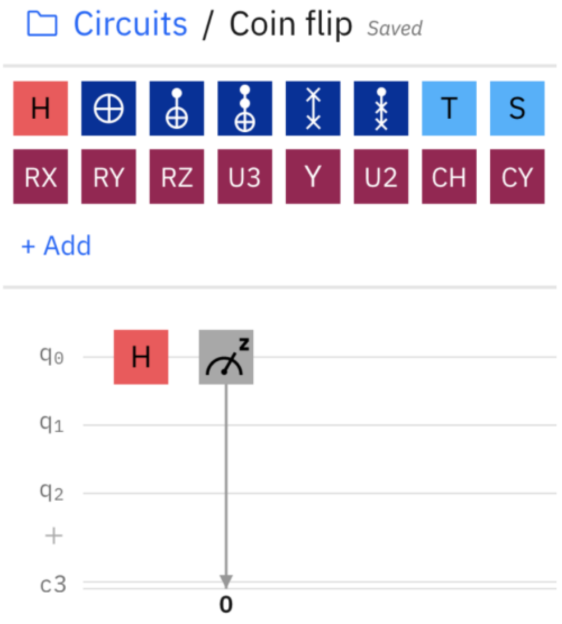


Figure 2.19 – Coin flip experiment

4. Name your circuit as `Coin flip` and save it.
5. Click **Run Settings** to expand the options.

6. Select the **ibmq_qasm_simulator** as the backend device and select the run count to **1024**. This will run the experiment 1,024 times.
7. Click **Run on ibmq_qasm_simulator**.
8. Once completed, click on the completed experiment in the **Completed jobs** list.

The results will now show two different states. Remember that the **Computational basis states** are represented along the *x* axis. The main difference you will now see is highlighted by the first classic bit of the experiment (the least significant bit on the far right of each state), which you can see is either a 0 or 1.

Another thing to note is the **Probabilities** (the *y* axis) of each of the two states. This will differ each time you run the experiment. For example, the results in the following screenshot will have a different result for the probability than your experiment:

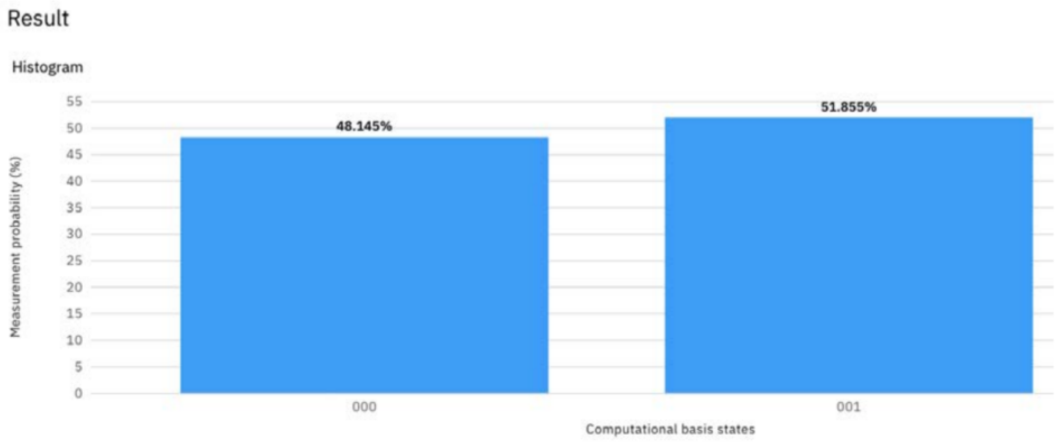


Figure 2.20 – Coin flip results

That being said, one thing you will notice from the preceding screenshot is that the results will fall fairly close to 50% each time you run the experiment. Rerun the experiment a few more times and examine the results for yourself.

The reason for this is our use of the Hadamard gate. This special gate leverages one of the two main quantum computing principles, **superposition**, that provides quantum computers with the potential to solve complex computations. We will cover what and how superposition works in *Chapter 4*, *Understanding Basic Quantum Computing Principles*, and how the Hadamard gate performs this gate operation on the qubit in *Chapter 6*, *Understanding Quantum Logic Gates*.

The use of the Hadamard gate, as you can see, allows your circuit to execute itself by leveraging a linear combination of two states, 0 and 1. As mentioned earlier, this helps to leverage superposition.

The second quantum computing principle used by quantum computers is **entanglement**. This quantum mechanical phenomenon helps us to entangle two or more qubits together. By entangling two qubits, we are in essence linking the value of one qubit and synchronizing it with another qubit. By synchronizing it, we mean that if I measure (observe) the value of one of the entangled qubits, then we can be sure that the other qubit will have the same value, whether you measure it at the same time or sometime later. The next experiment will cover this in more detail.

Entangling two coins together

Let's extend our coin-flipping example to include superposition by adding another coin and entangling them together so that when we run our experiment, we can determine the value of one coin without having to measure the other.

In the same way as our previous experiment, each qubit will represent a coin. In order to do this, we will use a multi-qubit gate called a **Control-Not** (**CNOT**) gate (pronounced *see-not*). The CNOT gate connects two qubits, where one is the source and the other the target. We will cover these gates in detail in *Chapter 6*, *Understanding Quantum Logic Gates*, but for now, here is a brief introduction so you can understand what you will expect to see.

When the source qubit (the qubit that is connected to the source of the CNOT gate) has a value equal to 1, then this enables the target of the CNOT, which as we can tell by the name is a NOT gate. This gate performs the same operation as the X gate that we ran in our previous first experiment, where we flipped the value of the qubit. Therefore, if the target qubit was set to 0, then it would flip the target qubit to 1 and vice versa. Let's try entangling our coins (qubits) to see how this works:

1. Open the Circuit Composer and create a new blank circuit.
2. Click and drag a Hadamard (H) gate onto the first qubit.
3. Click and drag the CNOT gate onto the first qubit (*round white gate with crosshairs on blue background*). This will drop the source onto the first qubit. When selecting the CNOT gate, the first qubit you drop it on will be set as the source. Visually, the source of the CNOT gate is a solid dot on the qubit to which the gate was dragged (see *Figure 2.21*).

By default, the target will set itself to the next qubit. In this case, it will drop to qubit 2. Visually, the target for a CNOT is a large dot with a cross in the middle, made to resemble a target.

4. Click and drag a measurement operator onto each of the two first qubits as shown in the following screenshot:

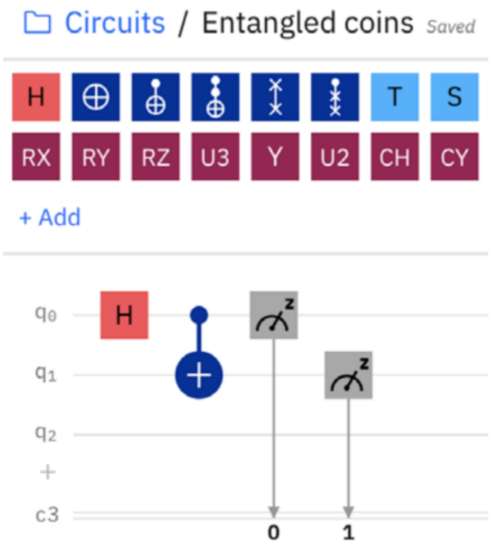


Figure 2.21 – Entangled qubit circuit representing entangled coins

5. Title and save your experiment as `Entangled coins`.
6. Click **Run Settings** on the circuit to launch the **Run Settings** dialog.
7. Select the **ibmq_qasm_simulator** or any other device from the backend selection as the backend device and select the run count to `1024`. This will run the experiment 1,024 times.
8. Click **Run on ibmq_qasm_simulator** (or whichever device you selected in the previous step).
9. Once completed click the **Coin flip** experiment from the **Completed jobs** list.

Now let's review the results and see what happens when we entangle two qubits:

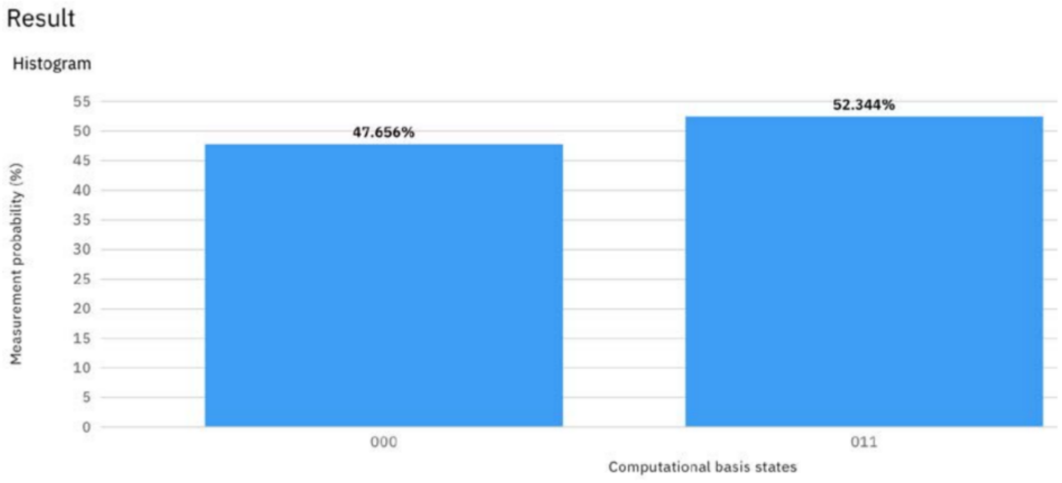


Figure 2.22 – Entangled coins results

As you can see in the preceding screenshot, the results still have two states, as they did in the previous experiment. However, one thing to observe here is the results of the two qubits. Note that the state of both qubits is either 000 or 011. Recall that the third bit (the most significant bit) was not operated on, so it remains in the initial state of 0.

What makes this experiment interesting is when we flipped one coin in the previous experiment, you saw that the results were 50% (0 or 1). However, now we are running the same experiment, but we are entangling another coin. In effect, this results in both coins becoming entangled together and thus their states will always be the same as each other. This means that if we flip both coins and we observe one of the coin values, then we know that the other entangled coin will be the same value.

Summary

In this chapter, you learned about the Circuit Composer view and its many components. You created three circuits. The first one was an experiment that simulated a classic NOT gate. The second one was an experiment in which a circuit was created using the Hadamard gate, which leveraged superposition. You then viewed the results of the experiment.

The third one was a circuit in which you expanded on the second circuit in order to include your first multi-gate, that is, a CNOT gate. From here, you demonstrated entanglement.

You were also able to review your results on a histogram, which allows you to examine how both superposition and entanglement results map from your quantum circuit to the classical bit outputs, as well as how to read the probabilities based on the results.

This has provided you with the skills to experiment with other gates and see what effect each operation has on each qubit and what information might be determined or used based on the results of the operation. This will be helpful when we look at some of the quantum algorithms and how these operations are leveraged to solve certain problems.

In the next chapter, we will move away from the click-and-drag work of the user interface and instead create experiments using Jupyter Notebooks, as well as beginning to program quantum circuits using Python.

Questions

1. Using the entangled coin-flip experiment, re-run the experiment. What is the statevector of the results?
2. What are the result states if you were to add a NOT gate before the Hadamard gate in the entangled coin-flip experiment's circuit?
3. Using the entangled coin-flip experiment from the Circuit Editor, switch the measurements so that the output of q_0 reads out to classic bit 1, and q_1 reads out to classic bit 0. What are the two states in the result and what are their probabilities?
4. What would the result states be if you were to add a Hadamard gate to the second qubit before the CNOT gate in the entangled coin-flip experiment's circuit?

3
Creating Quantum Circuits using Quantum Lab Notebooks

In this chapter, you will learn how to create circuits using the **Quantum Lab Notebooks** installed on the IBM Quantum Experience. You will learn how to save, import, and leverage existing circuits without having to install anything on your own computer. This will allow you to get a jump start on developing quantum circuits right away and ensure that you will be able to run the tutorials based on the currently installed version.

The following **Quantum Information Science Kit** (**Qiskit**) notebook topics will be covered in this chapter:

- Creating a quantum circuit using Quantum Lab Notebook
- Opening and importing existing Quantum Lab Notebook
- Developing a quantum circuit on Quantum Lab Notebook
- Reviewing results of your quantum circuit on the Quantum Lab Notebook

After completing this chapter, you will be able to leverage the capabilities of the Quantum Lab Notebooks, which will allow you to collaborate with others, share notebooks with others, import notebooks such as those that accompany this book, and run them directly from Quantum Lab. The Qiskit textbook is also capable of running on a Notebook, so as new features are released, you can be assured that you will be able to run them directly from your Notebook.

Technical requirements

In this chapter, some basic knowledge of programming is required, and some Python development is preferred. If you are familiar with other Notebook applications such as Jupyter Notebook then you may want to peruse this chapter, as most of the content here might be familiar to you.

We will not be using much Python-specific code here yet, but there will be some Qiskit code to help get you started in understanding and using the Qiskit Notebook. Here, I will cover the Qiskit basics as we go along, but rest assured we will have plenty of time in *Chapter 7*, *Introducing Qiskit and Its Elements* to review the many functions and features of Qiskit. Here is the source code used throughout this book: `https://github.com/PacktPublishing/Learn-Quantum-Computing-with-Python-and-IBM-Quantum-Experience`.

Here is the link for the CiA videos: `https://bit.ly/35o5M80`

Creating a quantum circuit using Quantum Lab Notebooks

Quantum Lab Notebooks provided to you via the IBM Quantum Experience platform will help you generate robust experiments that allow you to create quantum circuits and integrate those circuits with classical experiments or applications. Quantum Lab Notebooks generally contain a set of cells that you can use to write, test, and run your code in each cell individually.

You can also include **Markdown** language in the cells to capture any notes or non-code content, to help keep track of your learning or project. In this section, we will recreate the same quantum circuit you completed in *Chapter 2*, *Circuit Composer – Creating a Quantum Circuit*, only this time you will be using the Qiskit Notebook. So, let's get started!

Launching a Notebook from the Quantum Lab

To create a quantum circuit, let's start by launching the **Quantum Lab Notebook** from the **Quantum Lab** view. From the left panel under **Tools**, select **Quantum Lab** to launch the view, as illustrated in the following screenshot:

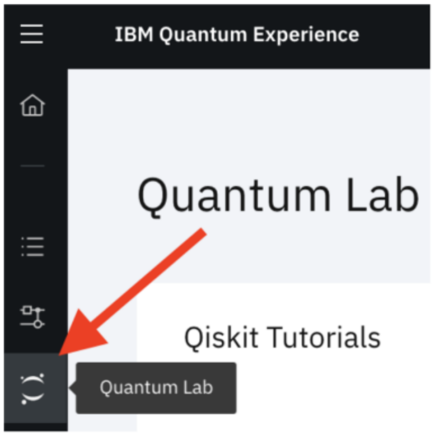


Figure 3.1 – Launching the Quantum Lab view (left panel)

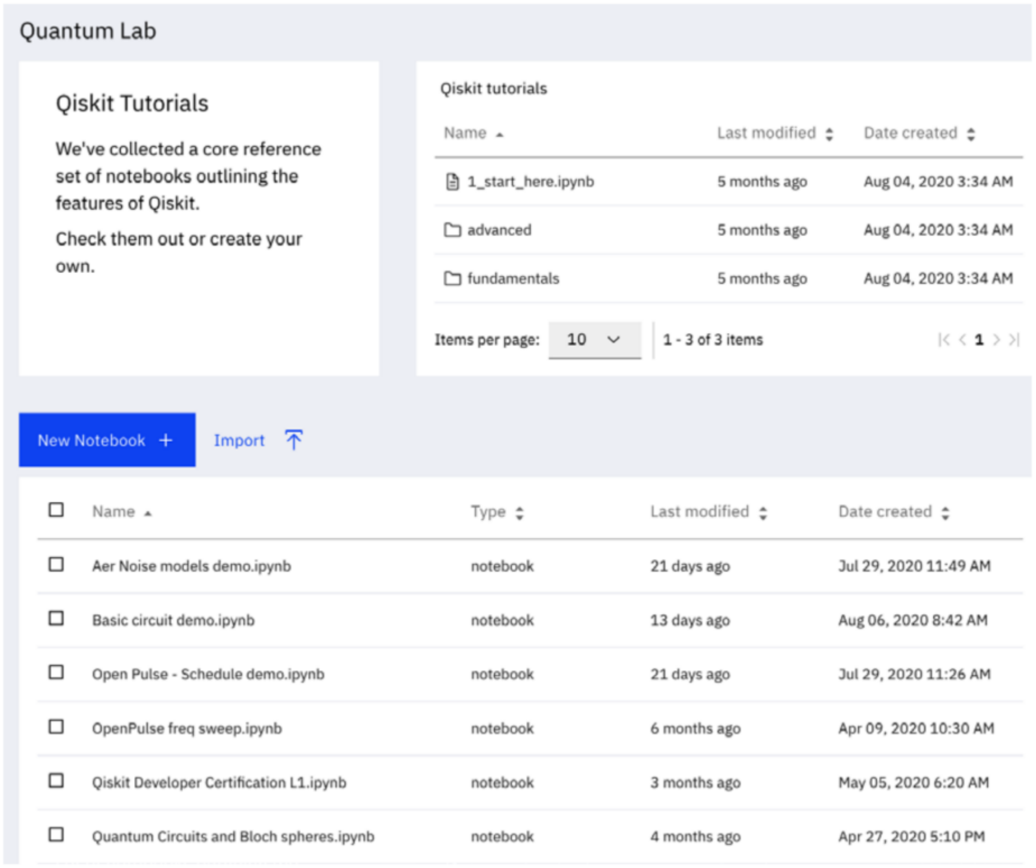
Now that you have the **Quantum Lab** view open, let's take a look at what each component of the Notebook provides.

Familiarizing yourself with the Quantum Lab components

In this section, we will become familiar with each of the components that make up the **Quantum Lab** view. As you see in *Figure 3.2* (starting from the top section, where you can see there are quick links to the **Qiskit tutorials**), the quick links are grouped into three sections, as follows:

- The first one is for starters, titled **1_start_here.ipynb**. This will review the introductory functions and features of Qiskit.
- The second group contains more **advanced** level tutorials.
- The third contains tutorials specific to certain **fundamentals** such as optimization, artificial intelligence, and many other domains.

Under the quick links is the list of all previously saved notebooks. You can choose to open any of those listed, or you can create or import notebooks by selecting either the **New Notebook +** or **Import** button respectively, as illustrated in the following screenshot:



Quantum Lab

Qiskit Tutorials

We've collected a core reference set of notebooks outlining the features of Qiskit.

Check them out or create your own.

Qiskit tutorials

Name	Last modified	Date created
1_start_here.ipynb	5 months ago	Aug 04, 2020 3:34 AM
advanced	5 months ago	Aug 04, 2020 3:34 AM
fundamentals	5 months ago	Aug 04, 2020 3:34 AM

Items per page: 10 | 1 - 3 of 3 items | 1

Name	Type	Last modified	Date created
Aer Noise models demo.ipynb	notebook	21 days ago	Jul 29, 2020 11:49 AM
Basic circuit demo.ipynb	notebook	13 days ago	Aug 06, 2020 8:42 AM
Open Pulse - Schedule demo.ipynb	notebook	21 days ago	Jul 29, 2020 11:26 AM
OpenPulse freq sweep.ipynb	notebook	6 months ago	Apr 09, 2020 10:30 AM
Qiskit Developer Certification L1.ipynb	notebook	3 months ago	May 05, 2020 6:20 AM
Quantum Circuits and Bloch spheres.ipynb	notebook	4 months ago	Apr 27, 2020 5:10 PM

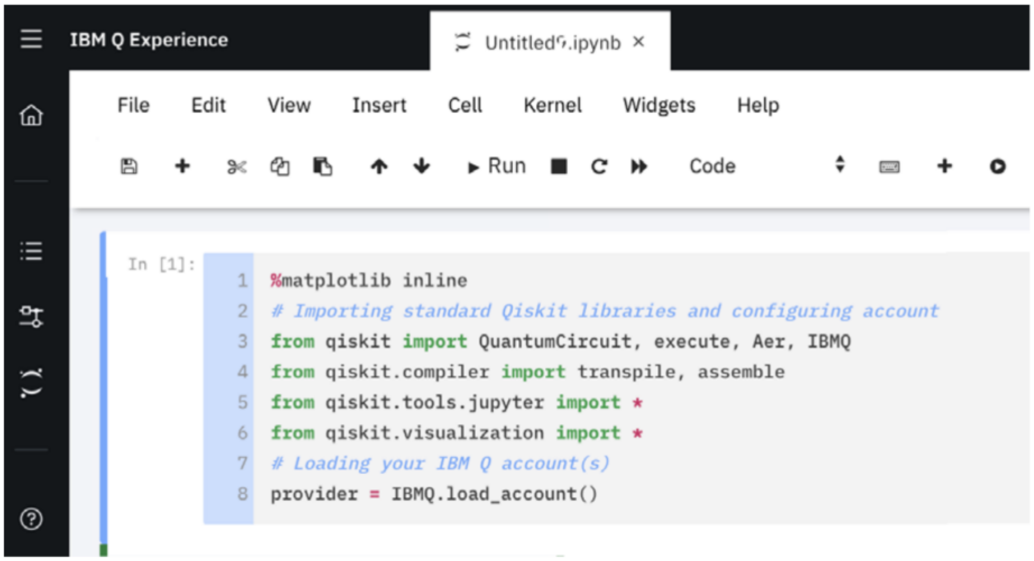
Figure 3.2 – Quantum Lab view

In the next step, we will create a new Notebook.

Creating a new Notebook

In this section, we will review the various functionalities available to ensure that you have a good understanding of all the different features available to you.

In the following screenshot, we can see the landing page of the **Circuit Composer** editor view:



```
%matplotlib inline
# Importing standard Qiskit libraries and configuring account
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

Figure 3.3 – Notebook landing page

The following points provide a description of the functions and features and what they contribute to the creation of a quantum circuit:

- When the Notebook loads up, you'll notice the first cell contains autogenerated code that includes some from Qiskit. Qiskit will be discussed in detail in *Chapter 7, Introducing Qiskit and its Elements*.

  The autogenerated code functions help to get your code up and running by adding some libraries and objects that are common when creating and running a quantum circuit. We'll review details of these objects further so that you can understand what each line pertains to and what the objects are generally used for.

  > **Important note**
  > Note that these may change as new features are added or updated to Qiskit, so the content of these lines may alter over time.

- The following lines of code, which can also be seen in the preceding screenshot, contain the most commonly used objects from the Qiskit library and the code for loading of your account details so that you can connect to the quantum systems.

  This is the first line of the autogenerated code block in your Notebook:

  ```
  %matplotlib inline
  ```

The preceding code imports the **Matplotlib** plotting library, which provides the ability to embed plots and publication quality figures into applications. Details about Matplotlib can be found on their home page here: `https://matplotlib.org/`

The next line imports four Qiskit objects that are commonly used to create and run a quantum circuit. `QuantumCircuit` is used to create a new circuit, which is a list of instructions bound to some registers. `execute` is an asynchronous call to run a circuit and return a job instance handle. `Aer` and `IBMQ` are providers for backend simulators and devices and to manage account details, respectively.

In the following code snippet, you can see we import each of these from the `qiskit` package:

```
from qiskit import QuantumCircuit, execute, Aer, IBMQ
```

`transpile` and `assemble` are compiler objects used to translate and compile circuits, while `assemble` provides a list of circuit schedules, as shown in the following code snippet:

```
from qiskit.compiler import transpile, assemble
```

- The following lines in the autogenerated block of code import all tools and objects from the **Jupyter Notebook** and visualization libraries, respectively. Jupyter Notebooks is what the Qiskit Notebook is built upon, so it will leverage existing features already familiar to those of you who compose experiments on a Jupyter Notebook.

  These features include creating new files, running kernels, and triggering cells. The visualization library includes many features used to visualize results from experiments such as histograms and bar charts, and in various formats, including Matplotlib, **Latex**, and so on. The code can be seen here:

```
from qiskit.tools.jupyter import *
%from qiskit.visualization import *
```

- And finally, the `IBMQ.load_account()` function loads your account information, particularly the **application programming interface** (**API**) token that was assigned to you when you initially registered. This is done if you desire to run an experiment on an actual device; the loading of your API token and other account information needed to run an experiment is already available without any extra work on your part. The following code snippet shows this:

```
provider = IBMQ.load_account()
```

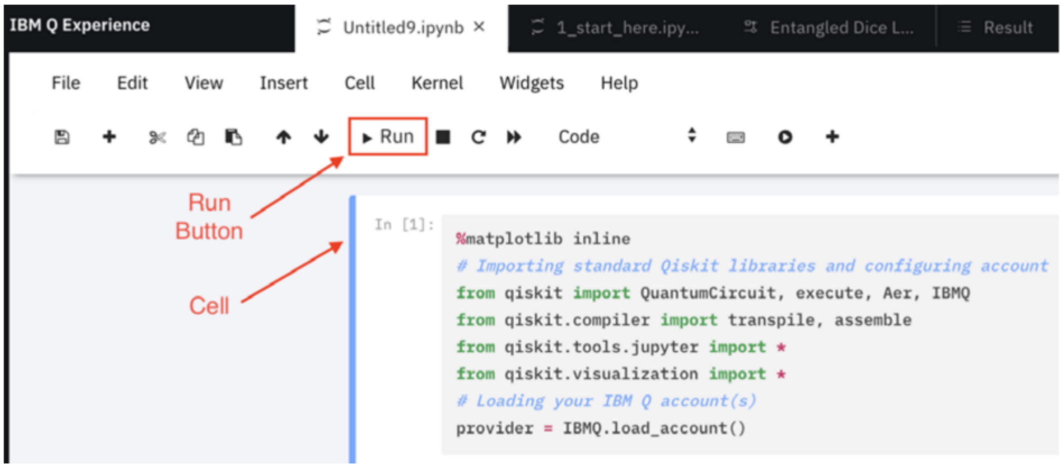
  This way, the content of your experiment is not cluttered with information that is not relevant to your experiment and conclusions.

Now that we are familiar with the autogenerated code, we'll take a quick look at the Qiskit Notebook itself. You'll note that its layout is very similar to that of a Jupyter Notebook, so those who are already familiar with Jupyter Notebooks will undoubtedly recognize the layout.

Learning about the Qiskit Notebook

For those who are new to the Qiskit Notebook, there are a couple of things to note that will help you understand how coding and running your code work. Those of you who are already familiar with Jupyter Notebooks can skip this section.

The Quantum Lab Notebooks run code one cell at a time. As shown in the following screenshot, a cell is a section of the Notebook that can contain text, metadata, and source code, such that it encapsulates the autogenerated code we looked at earlier. It simplifies coding by breaking up the code into these cells. The cells can be run individually by selecting the cell with your mouse and clicking the **Run** button, as illustrated in the following screenshot:



```
%matplotlib inline
# Importing standard Qiskit libraries and configuring account
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

Figure 3.4 – Notebook cell and operations

From the preceding screenshot, in the top row of the Qiskit Notebook menu options you will see some usual operations you would find in a typical document editor, as follows:

- **File**: **Save**, **Checkpoint**, **Print Preview**, **Close**, and **Halt**
- **Edit**: **Modify Cells**, **Move Cells**, **Merge Cells**, and so on
- **View**: **Toggle Headers**, **Toolbars**, and other views
- **Insert**: Insert cells above or below selected cell

- **Cell**: Run cell, run all cells, and more
- **Help**: Provides support content

There is one specific operation in the Notebooks menu list to take note of, and that is the **Kernel**. For those of you with an existing version(s) of Python, do take note that Qiskit, at the time of this writing, is running on **Python version 3**.

To confirm this, you can select **Kernel** from the drop-down menu and note that there is a **Change Kernel** option. You will see **Python 3** as the only option. However, if you install Qiskit on your local machine that contains other versions of Python, you might see them listed here as well. I mention this so as to ensure you have the correct kernel selected to run Qiskit experiments. Otherwise, you may encounter some errors due to version incompatibility.

Another thing to note is related to the various different formats in which you can download a Qiskit Notebook. By selecting the **File | Download as** option you will see the various formats, as shown in the following screenshot:

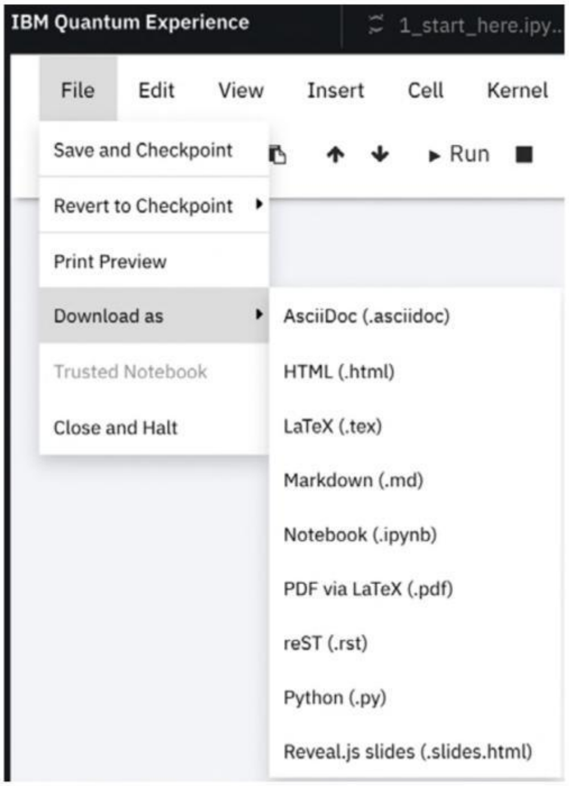
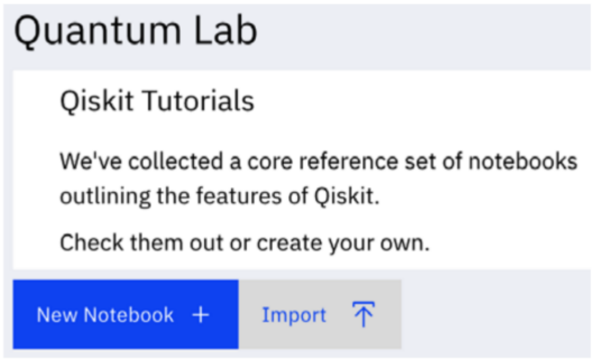


Figure 3.5 – Download formatting options

Up to now, you should be familiar with the functionality and features available on the Qiskit Notebook. These are features that make it easy to share experiments and make quick changes to them. We can now start creating and running quantum experiments on our notebooks using Qiskit.

Opening and importing existing Quantum Lab Notebook

Oftentimes, we wish to share our experiments with others and run others' experiments ourselves. Importing a Qiskit Notebook is easy in that the **Quantum Lab** home page has a link to **Import** button, next to the **New Notebook +** button, as shown in the following screenshot:


Quantum Lab

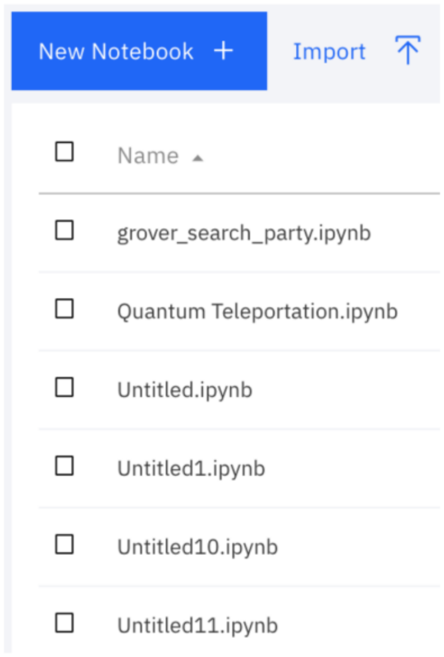
Qiskit Tutorials

We've collected a core reference set of notebooks outlining the features of Qiskit.

Check them out or create your own.

Figure 3.6 – Quantum Lab home page

This will launch your machine's file dialog to select the Qiskit Notebook you wish to import into your workspace on **IBM Quantum Experience** (**IQX**). To open an existing Qiskit Notebook, or one you have just imported, simply go back to your Qiskit Notebook page and select the Notebook you wish to open from the file list at the bottom of the view, as illustrated in the following screenshot:



- grover_search_party.ipynb
- Quantum Teleportation.ipynb
- Untitled.ipynb
- Untitled1.ipynb
- Untitled10.ipynb
- Untitled11.ipynb

Figure 3.7 – List of previously opened notebooks

Now that you are familiar with the Notebooks and the autogenerated code, let's quickly create the same circuit we generated in the previous chapter, only this time we will create it using only the Notebook.

Developing a quantum circuit on Quantum Lab Notebooks

Let's take a quick look at the quantum circuit we created in the previous chapter. For convenience, the circuit is given as follows:

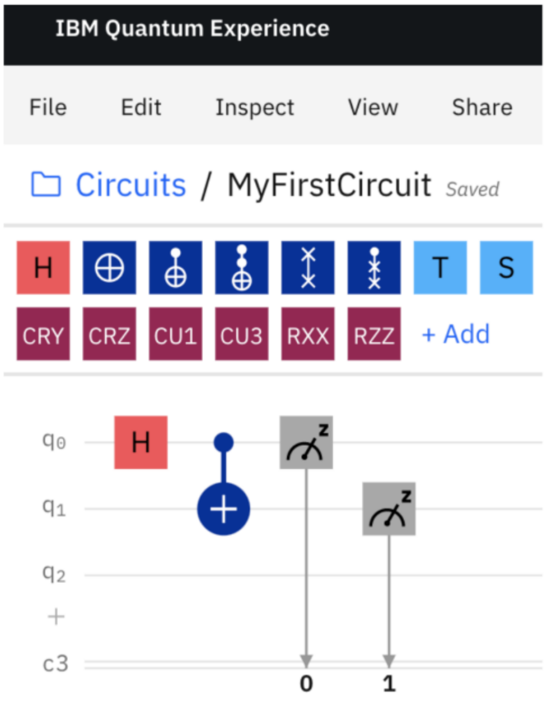


Figure 3.8 – Quantum circuit previously created using the Circuit Composer

The preceding circuit comprises of two gates—Hadamard and Controlled-Not—and two measurement operations on 2 qubits, respectively. This circuit was very easily constructed on the **Circuit Composer**; however, as we learn more about quantum circuits and begin to work on more complicated algorithms and circuits, it will be difficult to leverage a user interface such as the **Circuit Composer**. So, we will code the construction of quantum circuits and algorithms as we move forward instead.

To create the previous quantum circuit on a Notebook, follow these steps:

1. From the Quantum Lab Notebook, create a new Notebook and enter the following code in an empty cell:

```
qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0, 1)
```

   The preceding code creates `QuantumCircuit`. The two parameters pertain to the number of quantum bits (qubits) and classic bits we want to create, respectively. In this example, we will create two of each.

The second line adds a Hadamard gate onto the first qubit. Note that the index values of the qubits are `0` based. The third line adds a Controlled-Not gate that entangles the first qubit (q_0) as the control to the second qubit (q_1), the target. The parameters in the function pertain to the control and target, respectively.

2. Now, select **Run** to run the cell. Once the cell has completed running, you should see the output display `InstructionSet` of the results and a new cell generated below the one you just ran, as shown in the preceding code snippet. `InstructionSet` is a class of instruction collections and their contexts (classic and quantum arguments), where each context is stored separately per each instruction.

   Here is the output we are given after running the preceding code:

```
<qiskit.circuit.instructionset.InstructionSet at
0x7fc632176eb8>
```

3. Next, we will add the measurement operators to our circuit so that we can observe our results classically. Notice in the following code snippet that we are using the `range` method so as to simplify the mapping of each qubit to its respective classic bit:

```
qc.measure(range(2), range(2))
```

4. Run the preceding cell to include the measurement operators to our circuit. Now that we have created the circuit and included the same gates and operations we did in *Chapter 2, Circuit Composer – Creating a Quantum Circuit*, let's draw the circuit and compare. Draw the circuit using the `draw` function, shown as follows:

```
qc.draw()
```

   You should now see the following results, after the `draw` method is complete:

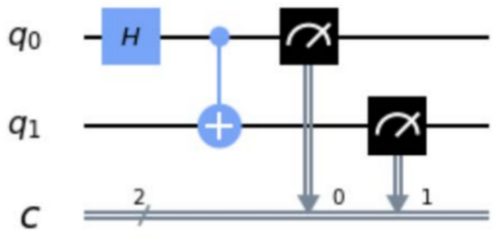


Figure 3.9 – Rendered image of the quantum circuit

Notice that the preceding circuit is identical to that which you created earlier. The only difference is the initial number of qubits. The **Circuit Composer** defaults to 5 qubits, whereas here we specified only 2. The circuit will run the same, both on the simulator and on an actual quantum device, since we are only using 2 qubits.

In this section, we have learned to navigate the Quantum Lab Notebook. We also learned how to open an existing Notebook, along with opening, creating, and importing the Notebook. We also saw how to develop a quantum circuit.

Now, we will move on to review the results of the quantum circuit.

Reviewing the results of your quantum circuit on Quantum Lab Notebooks

In this section, we'll conclude this chapter by running the circuit on a quantum simulator and a real device. We'll then review the results by following these steps:

1. From the open Notebook, enter and run the following in the next empty cell:

   ```
   backend = Aer.get_backend('qasm_simulator')
   ```

   The preceding code generates a `backend` object that will connect to the specified simulator or device. In this case, we are generating a `backend` object linked to the QASM simulator.

2. In the next empty cell, let's run the `execute` function. This function takes in three parameters—the circuit we wish to run, the backend we want to run it on, and how many `shots` we wish to execute. The returned object will be a job object with the contents of the executed circuit on the backend. The code for this can be seen here:

   ```
   job_simulator = execute(qc, backend, shots=1024)
   ```

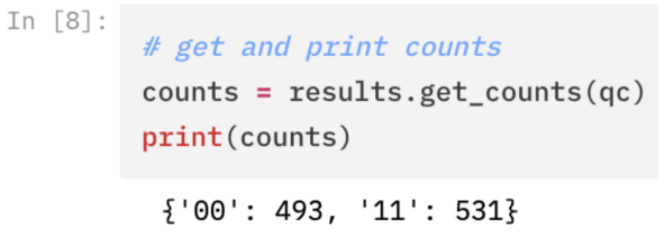
3. We now want to extract the results from the job object. In order to do that, we will call the `result` function, illustrated as follows, and save it in a new variable:

   ```
   result_simulator = job_simulator.result()
   ```

4. Since we ran our experiment with `1024` shots, we want to get the results from the counts. In order to do that, we can call the `get_counts()` method by passing in our circuit as the argument. Once we receive the counts, let's print out the results by running the following code:

   ```
   counts = result_simulator.get_counts(qc)
   print(counts)
   ```

   Note that the count results, shown as follows, may be different from your count results, which are based on the randomness of the qubits. But overall, the results will be similar by approximately 50%:



```
In [8]:  # get and print counts
         counts = results.get_counts(qc)
         print(counts)

         {'00': 493, '11': 531}
```

Figure 3.10 – Count results from the quantum circuit

5. Finally, let's visualize the result counts by plotting them using a histogram. We'll first import the `plot_histogram` method from the `qiskit.visualization` library and pass our `counts` in as an argument, as follows:

   ```
   from qiskit.visualization import plot_histogram
   plot_histogram(counts)
   ```

   As you can see in the following screenshot, the results are very similar to our results from the **Circuit Composer** in that 50% of the time our results are **00**, and the other half of the time they are **11**:

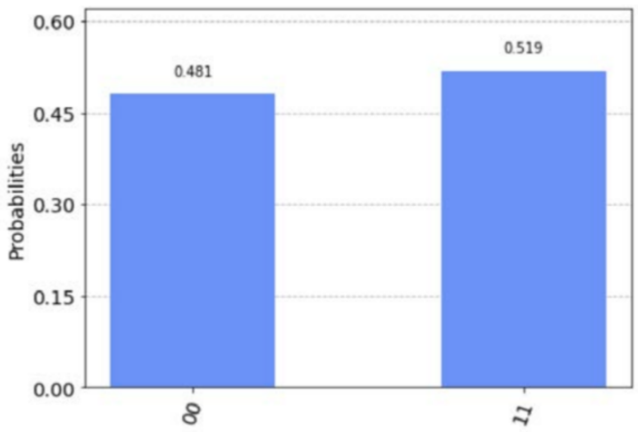


Figure 3.11 – Histogram view of the count results

Now that we have run our quantum circuit on a simulator, let's run this same quantum circuit on a real quantum computer.

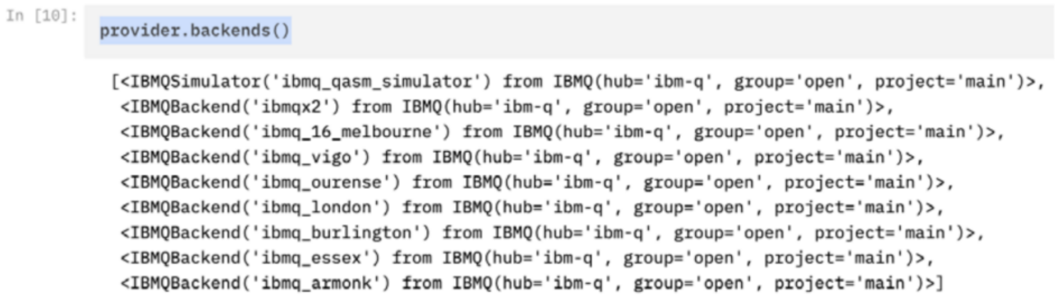
Executing a quantum circuit on a quantum computer

To run this quantum circuit on a quantum device, we will continue with the following steps:

1. The only change you need to update in the steps from the preceding section is to go from running on a simulator to a real device in *Step 1* from the previous steps, which is where you specify the name of the backend. In *Step 1*, we set the backend to the `qasm_simulator`. In this step, we will update to an actual device. So, let's first get a list of backends from our providers by running the following code in a new cell:

```
provider.backends()
```

The preceding method will return a list of all the simulators and real devices currently available to you, shown as follows. Note that the devices listed may change over time, so the results shown may be different when you run the method:



```
In [10]: provider.backends()
```

```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_london') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_burlington') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_essex') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

Figure 3.12 – List of available quantum computers (quantum devices)

2. The only change you need to update from the steps in the previous section is to specify which quantum computer from the list of backend devices you wish to run the experiment. In the previous steps, we set the backend to the `qasm_simulator`, whereas in this step we will update our backend to use a real device from the list. In this case, we'll choose `ibmq_vigo`. This list may appear different to you, so pick one from your list if `ibmq_vigo` is not listed. To do this, run the following code in a new cell:

```
backend = provider.get_backend('ibmq_vigo')
```

   The preceding code assigns the `ibmq_vigo` quantum computer as our backend.

3. From the previous steps, repeat *Step 2* to *Step 5* to run the circuit on a real device. Your results will seem a little different. Rather than just the **00** and **11** results, you will see that there are some **01** and **10** results, shown in the screenshot that follows, albeit only a small percentage of the time.

   This is due to noise from the real device, which is why they are often referred to as **Noisy Intermediate-Scale Quantum** (**NISQ**) systems or near-term devices. The noise can come from an array of things, such as ambient noise and decoherence. Details about the different types of noise and their effects will be discussed in detail in *Chapter 11*, *Mitigating Quantum Errors Using Ignis*.

The results can be seen in the following screenshot:

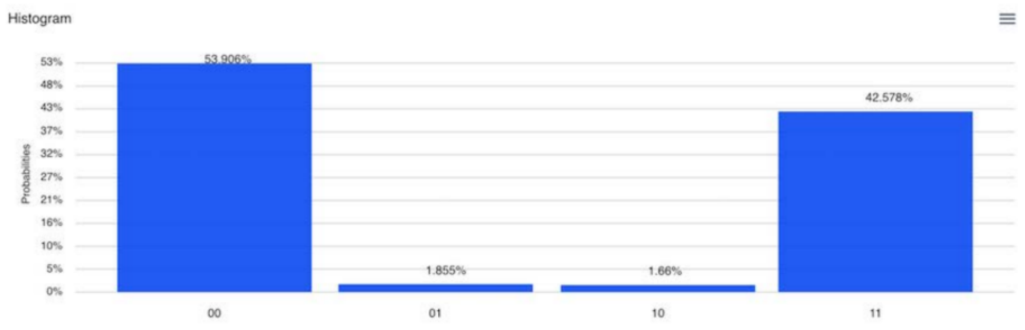


Figure 3.13 – Histogram plot of results

Congratulations! You have just completed running a quantum circuit on both a quantum simulator and a real quantum device using the Quantum Lab Notebooks. As you can see, by using the Notebook you can use many built-in Qiskit methods to create circuits and run them on various machines with a simple line of code, whereas on the **Circuit Composer** you would have to make various changes that would take a lot of time to complete.

Summary

In this chapter, you learned about the Quantum Lab Notebooks and ran a simple quantum circuit. You completed three basic functional steps: creating a quantum circuit using the Notebook and the Qiskit library, executing your circuit with a backend simulator and real device, and reviewing and visualizing your results from within the Notebook.

One thing you might have noticed is that using the Notebook with Qiskit also simplifies integrating your classical experiments with a quantum system. This has provided you with the skills and understanding to enhance your current Python experiments and run certain calculations on a quantum system, making them a hybrid classical/quantum experiment.

When the quantum calculations have completed, the results can be very easily used by your classical experiments.

Now that we are familiar with the Quantum Lab Notebooks and are able to create and execute a circuit, in the next chapter, we will start learning the basics of quantum computing and the quantum mechanical principles of superposition, entanglement, and interference.

Questions

1. Quantum Lab notebooks are built upon which application editor?
2. How would you create a 5-qubit circuit, as we did in *Chapter 2, Circuit Composer - Creating a Quantum Circuit*?
3. To run the experiment on another real device, which quantum computer would you select if your quantum circuit has more than 5 qubits?
4. When you run on a real device, can you explain why you get extra values when compared to running on a simulator?

Section 2: Basics of Quantum Computing

In this section, you will learn the basics needed to understand quantum computing, with particular focus on the mathematics and principles of quantum computing that most quantum algorithms leverage to potentially solve many intractable problems. You will also learn about the basic components, such as quantum bits, quantum gates, and quantum circuits, that we use to develop quantum algorithms.

This section comprises the following chapters:

4
Understanding Basic Quantum Computing Principles

Quantum computing, particularly its algorithms, leverage three quantum computing principles, namely, **superposition**, **entanglement**, and **interference**. In this chapter, we'll review each of these so that we can understand what each provides, the effect it has on each qubit, and how to represent them using the quantum gate sets provided to us. As a bonus, we will also create a quantum teleportation circuit that will leverage two of the three quantum computing principles to teleport an unknown state from one person to another.

The following topics will be covered in this chapter:

- Introducing quantum computing
- Understanding superposition
- Understanding entanglement
- Learning about the effects of interference between qubits
- Creating a quantum teleportation circuit

This chapter will focus on the three main quantum computing principles that will help you better understand how they are used in the various quantum algorithms. The quantum computers hosted on the **IBM Quantum Experience** leverage all these principles by use of the various quantum gates, some of which you used earlier in this book.

Technical requirements

In this chapter, some basic knowledge of programming is required. Some Python development knowledge is preferred as the experiments leverage Python libraries. Some general knowledge of physics is recommended; however, my goal is for the explanations to help you understand the quantum principles without the need for you to register for a physics course or read the **Feynman** lectures. Here is the full source code used throughout this book: `https://github.com/PacktPublishing/Learn-Quantum-Computing-with-Python-and-IBM-Quantum-Experience`.

Please visit the following link to check the CiA videos: `https://bit.ly/35o5M80`

Introducing quantum computing

Quantum computing isn't a subject that is as common as learning algebra or reading some of the literary classics. However, for most scientists and engineers or any other field that includes studying physics, quantum computing is part of the curriculum. For some of us who don't quite recall our studies in physics, or have never studied it, need not worry, as this section aims to provide you with information that will either refresh your recollection on the topic or at least perhaps help you understand what each of the principles used in quantum computing mean. Let's start with a general definition of quantum mechanics.

Quantum mechanics, as defined by most texts, is the study of nature at its smallest scale – in this case, the subatomic scale. The study of quantum mechanics is not new. Its growth began in the early 1900s by many physicists, whose names still chime in many of the current theories and experiments. The names of such physicists include Erwin Schrodinger, Max Plank, Werner Heisenberg, Max Born, Paul Dirac, and Albert Einstein, among others. As years passed, many other scientists expanded on the foundations of quantum mechanics and began performing experiments that would either prove, disprove, or oftentimes illustrate that there is no proof.

One of the more popular experiments is the **double slit** experiment. Although this is found in classical mechanics, it is referenced in quantum computing to describe the behavior of a quantum bit (qubit). It is in this experiment researchers were able to demonstrate that light (or photons) can be characterized as both waves and particles.

There were many distinct experiments that have been conducted over the years that illustrate this phenomenon, one of which was to fire particles through a double slit one at a time where at the other side of the double slit was a screen that captured, as a point, the location where each particle would hit. When only one slit was open, all the particles would appear as a stack of points directly in front of the slit, as shown in the following diagram:

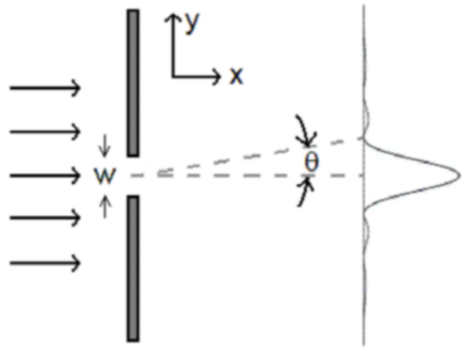


Figure 4.1 - Single-slit experiment (image source: `https://commons.wikimedia.org/wiki/File:SingleSlitDiffraction.GIF`)

From the previous diagram, you can see that all the particles are captured in an area directly across the slit.

However, when the second slit was open, it was imagined that there would be an identical stack of points on the screen. But this was not the case, as what was captured appeared to be a formation altogether different than what would be expected from a particle. In fact, it had the characteristics of a wave in that the points on the screen seemed to display a diffraction pattern, as shown in the following diagram:

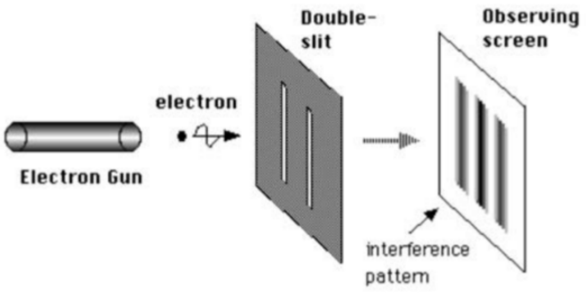


Figure 4.2 - Double-slit experiment (image source: `https://commons.wikimedia.org/wiki/File:Double-slit.PNG`)

From the previous diagram, you can see that all the particles are spread out from the center with interference gaps.

This diffraction pattern is caused by the interference of the light waves passing through the slits. Here, there are more points at the center of the screen than there are toward the outer ends of the observing screen. This wave particle phenomenon gave birth to lots of interesting research and development such as the **Copenhagen interpretation**, **many-worlds interpretation**, and the **De Broglie-Bohm** theory.

What this illustrated was that the light appeared as bands of light in certain areas of the board with some probability. By observing the preceding diagram, you can see that there is a higher probability that the electron fired from the gun will land in the center band of the screen as opposed to the outer bands. Also, note that due to interference, the spaces in between the bands that capture the electrons have less probability (blank areas between bands).

It is these effects of wave interference and probabilities that we will cover in this chapter, but first, we will start with the electron itself to understand superposition.

Understanding superposition

Superposition is something we generally can't see with the naked eye. This is typically the case when discussing the superposition of an electron. Since an electron is very small and there are so many of them, it is hard to distinguish one with even a powerful microscope. There are, however, some analogies in the classical world that we can use to illustrate what superposition is. For example, a spinning coin is what most texts use to describe superposition. While it is spinning, we can say that it is in the state of both heads and tails. It isn't until the coin collapses that we see what the final state of the coin is.

In this chapter, we're going to use this spinning coin analogy just to help you understand the general principle of superposition. However, once we start working on our quantum circuits, you will see some of the differences between superposition and its probabilistic behavior in the classical world versus its behavior in the quantum world. Let's start by reviewing the random effects in the classical world.

Learning about classical randomness

Previously, we discussed the randomness of a spinning coin as an example. However, the spinning coin and its results are not as random as we think. Just because we cannot guess the correct answer when a coin is spun on a table or flipped in the air does not make it random. What leads us to believe that it's random is the fact that we don't have all the information necessary to know or predict or, in fact, determine that the coin will land on either heads or tails.

All the relevant information, such as the weight of the coin, its shape, the amount of force required to spin the coin, the air resistance, the friction of the platform the coin is rolling on, and so on, all of this information, and the information of the environment itself, is not known to us in order for us to determine what the outcome would be after spinning a coin. It's because of this lack of information that we assume the spinning of the coin is random. If we had some function that could calculate all this information, then we would always successfully determine the outcome of the spinning coin.

The same can be said about random number generators. As an example, when we trigger a computer to generate a random number, the computer uses a variety of information to calculate and generate a random number. These parameters can include information such as the current daytime that the request was triggered, information about the user or the system itself, and so on.

These types of random number generators are often referred to as **pseudorandom number generators** (**PSRN**) or **deterministic random bit generators** (**DRBT**). They are only as random as the calculation or seed values provided that is allowed. For example, if we knew the parameters used and how they were used to generate this random number, then we would be able to determine the generated random number each and every time.

Now, that being said, I don't want you to worry about anyone determining the calculations or cryptic keys that you may have generated. We use these pseudorandom number generators because of the precision and granularity that they encompass to generate this number, which is such that any deviation can drastically alter the results.

So, why bother reviewing the probabilistic and random nature of a spinning coin? One, it's to explain the difference between randomness, or what we believe is random, in the classical world versus the randomness in the quantum world.

In the classic world, we learned that if we had all the information available, we can more than likely determine an outcome. However, in the previous section, where we described the double-slit experiment, we saw that we couldn't determine where in the screen the electron was going to hit. We understood the probabilities of where it would land based on our experiment. But even then, we could not deterministically identify where precisely the electron was going to land on the screen. You'll see an example of this when we create our superposition circuit in the next section.

For those who wish to learn a little more about this phenomenon, I would suggest reading the book by the famous physicist Richard Feynman titled *QED: The Strange Theory of Light and Matter*.

Preparing a qubit in a superposition state

In this section, we are going to create a circuit with a single qubit and set an operator on the qubit to set it in a superposition state. But before we do that, let's quickly define what a superposition state is.

We define the qubit as having two basis energy states, one of which is the ground (0) state and the second of which is the excited (1) state, as illustrated in *Figure 4.3*. The state value name of each basis state could be anything we choose, but since the results from our circuit will be fed back to a classic system, we will use binary values to define our states – in this case, the binary values 0 and 1. To say that the superposition of two states is *being in both 0 and 1 at the same time* is incorrect. The proper way to state a qubit is in a superposition state is to say that it is *in a complex linear combination of states where in this case, the states are 0 and 1.*

The following screenshot is referred to as a **Bloch sphere**, which represents a single qubit and its two basis states, which are located on opposite poles. On the north pole, we have the basis state 0, while the south pole, we have the basis state 1. The symbols surrounding the basis state values are the commonly used notations in most quantum computing text. This is called **Dirac notation**, which was named after the English theoretical physicist Paul Dirac, who first conceived the notation, which he called the **Bra-Ket notation**. Both Bra-Ket and Dirac notation are generally used interchangeably as they refer to the same thing, as we'll see later.

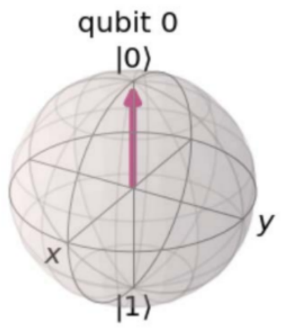


Figure 4.3 - Two basis states of a qubit on a Bloch sphere

Ok, so let's stop talking and let's start coding. We're going to create a quantum circuit with a single qubit. We will then execute the circuit so that we can obtain the same result we can see in the preceding screenshot, which is the initial state of the qubit, state $|0\rangle$.

Open a new Qiskit Notebook and enter the following code into the next empty cell:

```
from qiskit.visualization import plot_bloch_multivector
qc = QuantumCircuit(1)

# execute the quantum circuit
backend = Aer.get_backend('statevector_simulator')
result = execute(qc, backend).result()
stateVectorResult = result.get_statevector(qc)

#Display the Bloch sphere
plot_bloch_statevector(stateVectorResult)
```

The first line imports the Bloch sphere library so that we can plot our vector. The next line creates the circuit so that it includes `1` qubit, and in the next three lines, we are setting up our backend to execute the circuit to the simulator. And finally, we display the results on our Bloch sphere, which should display the same as what you can see in the preceding diagram.

So, you might be wondering what all this talk about vectors and statevector simulators is about. Good! This is what we will discuss now. The reason I wanted to run the experiment first as opposed to explaining what the vector states are and what the statevector simulator does is so that you can see it first and then hopefully the description will be a bit clearer. Let's start with the vector explanation.

Each qubit, as mentioned earlier, is made up of two basis states, which in this example reside on opposite poles of the Bloch sphere. These two basis states are what we would submit back to the classical system as our result – either one or the other. The vector representing these two points originates from the origin of the Bloch sphere, as you can see in the previous diagram, or the result from your experiment. If we were to notate this as a vector, we would write the following:

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

Since the opposite would apply to the opposite pole, we would notate it as follows:

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

From observing the vector values, you can see that flipping the values of the vector is similar to a classical bit flip. Now that we understand the vector representation of a qubit, let's continue and set the qubit in a superposition state:

1. Insert a new cell at the bottom of the current notebook and enter the following code:

```
#Place the qubit in a superposition state by adding a
#Hadamard (H)gate
qc.h(0)
#Draw the circuit
qc.draw()
```

The previous code places a Hadamard (H) gate onto the first qubit, identified by the qubit's index value (`0`). It then calls the `draw` function, which will draw the circuit diagram.

After running the previous cell, you should see the following circuit image, which represents adding the Hadamard gate to the qubit:

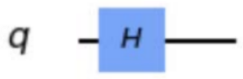


Figure 4.4 – Circuit with a Hadamard (H) gate added to a qubit

The **Hadamard gate** is an operational gate that places the qubit in a superposition state, or, more specifically, a complex linear combination of the basis states, which means that when we measure the qubit, it will have an equal probability result of measuring a 0 or 1. Or in other words, it would collapse to the basis state value $|0\rangle$ or $|1\rangle$.

Mathematically, the superposition state is represented in the following two superposition equations, which, as you can see, depends on which of the two basis states it was in prior to applying the Hadamard gate. The first superposition equation is as follows and originates from the $|0\rangle$ state:

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

The second superposition equation, originating from the $|1\rangle$ state, is as follows:

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

This is equal to a $\pi/2$ rotation about the X and Z axes of the Bloch sphere. These rotations are Cartesian rotations, which rotate counter-clockwise.

2. Now, let's execute our circuit, see what this looks like, and where the state vector lands on the Bloch sphere. In the following code, you will execute the same circuit again, the results of which will not differ in that the qubit will appear in a superposition state, which you will see in the resulting Bloch sphere's output:

```
#Execute the circuit again and plot the result in the
#Bloch sphere
result = execute(qc, backend).result()
#Get the state vector results of the circuit
stateVectorResult = result.get_statevector(qc)

#Display the Bloch sphere
plot_bloch_multivector(stateVectorResult)
```

Once the circuit has completed executing, the results will be plotted on the Bloch sphere in a superposition between $|0\rangle$ and $|1\rangle$, as illustrated in the following screenshot:

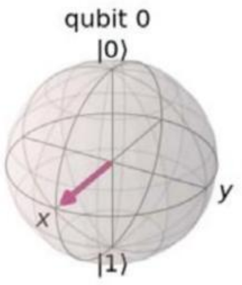


Figure 4.5 - Superposition of a qubit after 90° rotation around the X and Z axes

As you can see in the preceding screenshot, this has placed the vector on the positive X axis, as described previously when adding a H gate from the $|0\rangle$ basis state.

3. Now, let's clear the circuit. This time, we will initialize the qubit to the $|1\rangle$ state first and then apply a Hadamard gate to see what happens to the vector this time. Initialize qubit to the $|1\rangle$ state and place it in a superposition. Clear the circuit and initialize `qubit` to `1` before applying `Hadamard gate`:

```
#Reset the circuit
qc = QuantumCircuit(1)
#Rotate the qubit from 0 to 1 using the X (NOT) gate
```

In this section, you will learn about the community, its many programs, and how you can contribute and become a **Qiskit Advocate**. Qiskit Advocates are members of the Qiskit community who have passed a rigorous exam, have made many contributions to the Qiskit community, and who have helped many others along the way. Let's start by introducing you to the community itself.

Introducing the Qiskit community

Ever since Qiskit was first deployed as an open source project, the open source community has contributed so many features and enhancements that it has only improved over time. The development ecosystem itself has flourished so much that it is being used in universities, industry, and governments around the world, even in Antarctica!

Members of the Qiskit community, often referred to as **Qiskitters**, often work together as a solid diverse group to ensure everyone is supported. Whether they are newbies to quantum computing or veteran quantum researchers, they all share a passion for collaborating and connecting on various projects. The link to the Qiskit community is `https://www.qiskit.org/education`.

One of the early projects was to create resources for those new to quantum computing. These resources vary from generating enablement materials to **YouTube** video series. The topics included both hardware and software, which describes what happens on the backend, to software that describes new research that others are working on. Along with the resources, there are also events that are planned all over the world at any given time. This includes events such as workshops, where communities join either in person or virtually in order to learn the latest in quantum computing.

Other events also include **hackathons** and code camps, of which the largest is the **Qiskit Camp**, which the IBM Quantum team hosts quarterly in different continents around the world. The 3-to-4-day camp usually includes accommodations in very exotic locations, meals, transportation to and from airports, and so on. Researchers from **IBM Research** also participate as lecturers, coaches, and judges. Teams are created and brainstorm ideas for projects that they work together on during the weekend, where they then have the opportunity to compete and win prizes. This is very similar to hackathons.

Recently, the Qiskit community initiated the **Qiskit Advocate program**. This program was created to provide support to individuals who have actively been involved with the Qiskit community and have contributed over time. To become a Qiskit Advocate, you would need to apply online (`https://qiskit.org/advocates`), where you will be given an exam to test your knowledge of Qiskit and specify at least three community contributions. These qualifications, of course, can change over time, so it is recommended that you check the site for any updates and application deadlines.

The test covers all four elements and some quantum computing knowledge. Besides knowledge and a passing score, the Qiskit Advocate candidate must also have contributed to the Qiskit community. This can be done in a variety of ways, such as contributing to the Qiskit open source code and supporting other community members by either providing assistance or creating educational material that helps others learn about quantum computing and Qiskit.

Once accepted into the Qiskit Advocate program, you will have the opportunity to network with other experts and access core members of the Qiskit development team. You will also gain support and recognition from IBM through the Qiskit community, as well as receive invitations to special events such as Qiskit Camp, hackathons, and other major events where you can not only collaborate with others but lead or coach as well.

Contributing to the Qiskit community

Support across members is key, not just for Qiskit Advocates but for all members. The Qiskit community has set up various channels to offer support to all the members of the community. They have a **Slack workspace** (`http://ibm.co/joinqiskitslack`) that is very active and has various channels so that members can ask questions, post event updates, or just chat about the latest quantum research that had been recently published. There are also other collaborative sources that Qiskit connects through. The current list of collaboration tools can be found at the bottom of the main Qiskit page: `https://www.qiskit.org/`.

Specializing your skill set in the Qiskit community

One of the most common questions asked about contributing to the Qiskit community, particularly those who are interested in becoming Qiskit Advocates, is, *what are the various ways you can contribute?* There are many ways in which you can contribute to the Qiskit community. Ideally, you want to become familiar with the different forms of contributions, such as the following:

- **Code contributions**: Adding a new feature, optimizing the performance of a function, and bug fixes are some of the good ways to start if you are a developer. If you are new to coding, there is a label that the Qiskit development team has created for this called **good first issue**. This is an umbrella term for the issues that are ideal for those who are new to the code base.

- **Host a Qiskit event in your area or virtually**: You can host an event and invite a Qiskit Advocate to run a workshop or talk to a group about the latest updates in Qiskit.
- **Help others**: You can help others by answering questions asked by other community members, reporting bugs, identifying features that may enhance the development of circuits, and so on.

Specializing in an area such as noise mitigation, error correction, or algorithm design is an advantage to the community. The **Qiskit Slack community** has a number of channels that focus on specific areas of quantum computing, including each of the four elements, quantum systems, quantum experience, Qiskit Pulse, Qiskit on Raspberry Pi, and many more. If you specialize in any of these areas, you can join the Slack group and collaborate on the many technologies and topics.

In this section, you learned about the open source contribution process and how to find tasks for starters and experts so that everyone can contribute.

Summary

In this chapter, you learned about the general features and capabilities provided by each of the four Qiskit elements so that you can create highly efficient quantum algorithms. You then learned how to install Qiskit locally, as well as how to contribute and find support from the Qiskit community.

Out of the four Qiskit elements available, we learned about Terra first. This provided you with the skills and functionality to create circuits, and you then applied these operations to the qubits via gates and operators.

Then, we learned about Aer, which allows us to create better simulators, and Ignis, which helps us mitigate errors and calculate the quantum volume of a system.

After that, we learned about Aqua. We understood that it is generally a high-level view of quantum computing that eliminates a lot of the underlying details of building a circuit and mitigating noise and errors. This helps simplify integrating your classical applications into a quantum system by leveraging the many quantum and classical algorithms available. Then, we learned about Qiskit community support and its advantages to all, particularly those who are new to quantum computing and need a little support to understand some of the challenging content.

Index