*Learn Quantum Computing with Python and Q#*

Copyrighted material

# *Learn Quantum Computing with*
# *thon and Q#*

Copyrighted image

## A HANDS-ON APPROACH

SARAH KAISER
AND CHRIS GRANADE

MANNING
SHELTER ISLAND

Copyrighted material

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

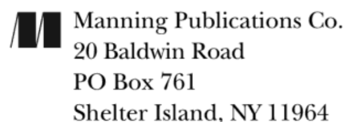
Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: orders@manning.com

©2021 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

♾ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.



Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964

Development editor: Dustin Archibald
Technical development editors: Alain Couniot and Joel Kotarski
Review editor: Ivan Martinović
Production editor: Deirdre S. Hiam
Copy editor: Tiffany Taylor
Proofreader: Katie Tennant
Technical proofreader: Krzysztof Kamyczek
Typesetter: Dennis Dalinnik
Cover designer: Marija Tudor

ISBN: 9781617296130
Printed in the United States of America

Copyrighted material

# brief contents

PART 1 GETTING STARTED WITH QUANTUM .................................. 1

1 ■ Introducing quantum computing 3
2 ■ Qubits: The building blocks 17
3 ■ Sharing secrets with quantum key distribution 54
4 ■ Nonlocal games: Working with multiple qubits 75
5 ■ Nonlocal games: Implementing a multi-qubit simulator 90
6 ■ Teleportation and entanglement: Moving quantum data around 108

PART 2 PROGRAMMING QUANTUM ALGORITHMS IN Q# ............ 131

7 ■ Changing the odds: An introduction to Q# 133
8 ■ What is a quantum algorithm? 152
9 ■ Quantum sensing: It's not just a phase 185

PART 3 APPLIED QUANTUM COMPUTING .................................. 217

10 ■ Solving chemistry problems with quantum computers 219
11 ■ Searching with quantum computers 249
12 ■ Arithmetic with quantum computers 278



Copyrighted material

Copyrighted material

# contents

*foreword* *xv*
*preface* *xvii*
*acknowledgments* *xix*
*about this book* *xxi*
*about the authors* *xxv*
*about the cover illustration* *xxvi*

PART 1 GETTING STARTED WITH QUANTUM ....................1

1 ***Introducing quantum computing*** ***3***

1.1 Why does quantum computing matter? 4
1.2 What is a quantum computer? 5
1.3 How will we use quantum computers? 8
*What can quantum computers do?* *10* ▪ *What can't quantum computers do?* *11*
1.4 What is a program? 13
*What is a quantum program?* *14*



Copyrighted material

2 Qubits: The building blocks 17
2.1 Why do we need random numbers? 19
2.2 What are classical bits? 22
*What can we do with classical bits? 23 ▪ Abstractions are our friend 26*
2.3 Qubits: States and operations 27
*State of the qubit 28 ▪ The game of operations 30 Measuring qubits 34 ▪ Generalizing measurement: Basis independence 38 ▪ Simulating qubits in code 41*
2.4 Programming a working QRNG 46

3 Sharing secrets with quantum key distribution 54
3.1 All's fair in love and encryption 54
*Quantum NOT operations 58 ▪ Sharing classical bits with qubits 62*
3.2 A tale of two bases 63
3.3 Quantum key distribution: BB84 66
3.4 Using a secret key to send secret messages 71

4 Nonlocal games: Working with multiple qubits 75
4.1 Nonlocal games 76
*What are nonlocal games? 76 ▪ Testing quantum physics: The CHSH game 76 ▪ Classical strategy 80*
4.2 Working with multiple qubit states 81
*Registers 81 ▪ Why is it hard to simulate quantum computers? 83 ▪ Tensor products for state preparation 85 Tensor products for qubit operations on registers 86*

5 Nonlocal games: Implementing a multi-qubit simulator 90
5.1 Quantum objects in QuTiP 91
*Upgrading the simulator 96 ▪ Measuring up: How can we measure multiple qubits? 99*
5.2 CHSH: Quantum strategy 103

Copyrighted material

6 *Teleportation and entanglement: Moving quantum data around* 108

6.1 Moving quantum data 109

*Swapping out the simulator 112 ▪ What other two-qubit gates are there? 115*

6.2 All the single (qubit) rotations 117

*Relating rotations to coordinates: The Pauli operations 119*

6.3 Teleportation 126

PART 2 PROGRAMMING QUANTUM ALGORITHMS IN Q# ...131

7 *Changing the odds: An introduction to Q#* 133

7.1 Introducing the Quantum Development Kit 134

7.2 Functions and operations in Q# 137

*Playing games with quantum random number generators in Q# 138*

7.3 Passing operations as arguments 143

7.4 Playing Morgana's game in Q# 149

8 *What is a quantum algorithm?* 152

8.1 Classical and quantum algorithms 153

8.2 Deutsch–Jozsa algorithm: Moderate improvements for searching 156

*Lady of the (quantum) Lake 156*

8.3 Oracles: Representing classical functions in quantum algorithms 161

*Merlin's transformations 162 ▪ Generalizing our results 165*

8.4 Simulating the Deutsch–Jozsa algorithm in Q# 170

8.5 Reflecting on quantum algorithm techniques 174

*Shoes and socks: Applying and undoing quantum operations 175*
*Using Hadamard instructions to flip control and target 178*

8.6 Phase kickback: The key to our success 180

Copyrighted material

*9 Quantum sensing: It's not just a phase 185*

9.1 Phase estimation: Using useful properties of qubits for measurement 186

*Part and partial application 186*

9.2 User-defined types 191

9.3 Run, snake, run: Running Q# from Python 197

9.4 Eigenstates and local phases 202

9.5 Controlled application: Turning global phases into local phases 206

*Controlling any operation 210*

9.6 Implementing Lancelot's best strategy for the phase-estimation game 212

9.7 Summary 215

9.8 Part 2: Conclusion 215

PART 3 APPLIED QUANTUM COMPUTING ........................ 217

*10 Solving chemistry problems with quantum computers 219*

10.1 Real chemistry applications for quantum computing 220

10.2 Many paths lead to quantum mechanics 222

10.3 Using Hamiltonians to describe how quantum systems evolve in time 225

10.4 Rotating around arbitrary axes with Pauli operations 229

10.5 Making the change we want to see in the system 237

10.6 Going through (very small) changes 239

10.7 Putting it all together 242

*11 Searching with quantum computers 249*

11.1 Searching unstructured data 250

11.2 Reflecting about states 256

*Reflection about the all-ones state 257 ▪ Reflection about an arbitrary state 258*

11.3 Implementing Grover's search algorithm 264

11.4 Resource estimation 271

Copyrighted material

# *foreword*

For most of its history, quantum computing was a field for physicists—perhaps a few having a proclivity for computer science, but not necessarily so. The popular textbook, *Quantum Computation and Quantum Information*, by Michael A. Nielsen and Isaac L. Chuang, is still considered the go-to textbook, and was written by two quantum physicists. To be sure, computer scientists have always been around, but some theoreticians wear how few lines of code they have written as a badge of honor. This is the quantum world myself, Kaiser, and Granade came of age in. I could easily shake my fist at the new cohort of students and yell, "When I was your age, we didn't write code—we choked on chalk dust!"

I met Chris Granade while we were both graduate students. Back then we wrote academic journal articles for physics journals that contained lines of code which were rejected for being "not physics." But we were not deterred. And now, many years later, this book represents for me the ultimate vindication! This is a book that teaches you everything you'll ever want and need to know about quantum computing, without the need for physics—though, if you really want to know the connection back to physics, Kaiser and Granade offer that as well 😆? There are also emojis 😊!

I've come a long way since then, and I owe much to Granade, as does the field of quantum computing, for showing many of us that between the "quantum" and the "computing," there is more than just theorems and proofs. Kaiser has also taught me more than I thought existed about the need for the software engineer's touch in developing quantum technology. Kaiser and Granade have turned their expertise into words and lines of code so all can benefit from it, as I have.



Copyrighted material

Though the goal was to create "not a textbook," this book could certainly be used as such in a university lecture as introductions to quantum computing shift from physics departments to schools of computer science. There is immense growing interest in quantum computing, and the majority of it is not coming from physics—software developers, operations managers, and financial executives all want to know what quantum computing is about and how to get their hands on it. Gone are the days of quantum computing as a purely academic pursuit. This book serves the needs of the growing quantum community.

Though I've alluded to the decreasing proportion of physicists in the field of quantum computing, I don't want to discount them. Just as I was once a software development Luddite, this book is really for anyone—especially those already in the field who want to learn about the software side of quantum computing in a familiar setting.

Fire up your favorite code editor and get ready to print ("Hello quantum world!").

CHRIS FERRIE, PhD
Associate Professor, Centre for Quantum Software and Information
Sydney, NSW, Australia

Copyrighted material

# *preface*

Quantum computing has been our jam for more than 20 years combined, and we are passionate about taking that experience and using it to help more folks get involved in quantum technologies. We completed our doctoral degrees together, and while doing so, we struggled through research questions, pun competitions, and board games, helping to push the boundaries of what was possible with qubits. For the most part, this meant developing new software and tools to help us and our teams do better research, which was a great bridge between the "quantum" and "computing" parts of the subject. However, while developing various software projects, we needed to teach our developer colleagues what we were working on. We kept wondering, "Why isn't there a good book for quantum computing that's technical but not a textbook?" What you are currently looking at is the result. 💖

We've written the book to be accessible to developers, rather than writing it in the textbook style that is so typical in other quantum computing books. When we were learning quantum computing ourselves, it was very exciting but also a bit scary and intimidating. It doesn't have to be that way, as a lot of what makes quantum computing topics confusing is the *way* they are presented, not the content.

Unfortunately, quantum computing is often described as "weird," "spooky," or beyond our understanding, when the truth is that quantum computing has become quite well understood during its 35-year history. Using a combination of software development and math, *you* can build up the basic concepts you need to make sense of quantum computing and explore this amazing new field.



Copyrighted material

Our goal with this book is to help *you* learn the basics about the technology and equip you with tools you can use to build the quantum solutions of tomorrow. We focus on hands-on experience with developing code for quantum computing. In part 1, you'll build your own quantum device simulator in Python; in part 2, you'll learn how to apply your new skills to writing quantum applications with Q# and the Quantum Development Kit; and in part 3, you'll learn to implement an algorithm that factors integers exponentially faster than the best-known conventional algorithm—and throughout, *you* are the one doing it, and this is *your* quantum journey.

We have included as many practical applications as we can, but the truth is, that's where you come in! Quantum computing is at a cusp where to go forward, we need a bridge between the immense amount that's known about what quantum computers can and can't do and the problems that people need to solve. Building that bridge takes us from quantum algorithms that make for great research to quantum algorithms that can impact all of society. You can help build that bridge. Welcome to your quantum journey; we're here to help make it fun!

Copyrighted material

# *acknowledgments*

We didn't know what we were getting into with this book at the start; all we knew was that a resource like this needed to exist. Writing the book gave us a huge opportunity to refine and develop our skills in explaining and teaching the content we were familiar with. All the folks we worked with at Manning were wonderful—Deirdre Hiam, our production editor; Tiffany Taylor, our copyeditor; Katie Tennant, our proofreader; and Ivan Martinović, our reviewing editor—and they helped us make sure this was the best book it could be for our readers.

We thank Olivia Di Matteo and Chris Ferrie for all of their valuable feedback and notes, which helped keep our explanations both accurate and clear.

We also thank all the reviewers of the manuscript who looked at it in various stages of development and whose thoughtful feedback made this a much better book: Alain Couniot, Clive Harber, David Raymond, Debmalya Jash, Dimitri Denisjonok, Domingo Salazar, Emmanuel Medina Lopez, Geoff Clark, Javier, Karthikeyarajan Rajendran, Krzysztof Kamyczek, Kumar Unnikrishnan, Pasquale Zirpoli, Patrick Regan, Paul Otto, Raffaella Ventaglio, Ronald Tischliar, Sander Zegveld, Steve Sussman, Tom Heiman, Tuan A. Tran, Walter Alexander Mata López, and William E. Wheeler.

Our thanks go to all of the Manning Early Access Program (MEAP) subscribers who helped find bugs, typos, and places to improve the explanations. Many folks also provided feedback by filing issues on our sample code repo: our thanks go to them!

We would like to acknowledge the many great establishments around the Seattle area (notably Caffe Ladro, Miir, Milstead & Co., and Downpour Coffee Bar) that tolerated us drinking coffee after coffee and talking animatedly about qubits, as well as the



Copyrighted material

## Who should read this book

This book is intended for people who are interested in quantum computing and have little to no experience with quantum mechanics but do have some programming background. As you learn to write quantum simulators in Python and quantum programs in Q#, Microsoft's specialized language for quantum computing, we use traditional programming ideas and techniques to help you out. A general understanding of programming concepts like loops, functions, and variable assignments will be helpful.

Similarly, we use some mathematical concepts from linear algebra, such as vectors and matrices, to help us describe quantum concepts; if you're familiar with computer graphics or machine learning, many of the concepts are similar. We use Python to review the most important mathematical concepts along the way, but familiarity with linear algebra will be helpful.

## How this book is organized: A roadmap

This text aims to enable you to start exploring and using practical tools for quantum computing. The book is broken into three parts that build on each other:

- Part 1 gently introduces the concepts needed to describe *qubits*, the fundamental unit of a quantum computer. This part describes how to simulate qubits in Python, making it easy to write simple quantum programs.
- Part 2 describes how to use the Quantum Development Kit and the Q# programming language to compose qubits and run quantum algorithms that perform differently from any known classical algorithms.
- In part 3, we apply the tools and methods from the previous two parts to learn how quantum computers can be applied to real-world problems such as simulating chemical properties.

There are also four appendixes. Appendix A has all the installation instructions for setting up the tools we use in the book. Appendix B is a quick reference section with a quantum glossary, notation reminders, and code snippets that may be helpful as you progress through the book. Appendix C is a linear algebra refresher, and appendix D is a deep dive into one of the algorithms you will be implementing.

## About the code

All the code used in this book can be found at https://github.com/crazy4pi314/learn-qc-with-python-and-qsharp. Full installation instructions are available at the repository for this book and in appendix A.

The book's samples can also be run online without installing anything, using the mybinder.org service. To get started, go to https://bit.ly/qsharp-book-binder.

Copyrighted material

## liveBook discussion forum

Purchase of *Learn Quantum Computing with Python and Q#* includes free access to a private web forum run by Manning Publications where you can make comments about the book, ask technical questions, and receive help from the authors and from other users. To access the forum, go to https://livebook.manning.com/#!/book/learn-quantum-computing-with-python-and-q-sharp/discussion. You can also learn more about Manning's forums and the rules of conduct at https://livebook.manning.com/#!/discussion.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and authors can take place. It is not a commitment to any specific amount of participation on the part of the authors, whose contribution to the forum remains voluntary (and unpaid).We suggest you try asking them some challenging questions lest their interest stray! The forum and the archives of previous discussions will be accessible from the publisher's website as long as the book is in print.

## Other online resources

As you start your quantum computing journey by reading this book and working through the provided sample code, you may find the following online resources helpful:

- *Quantum Development Kit documentation* (https://docs.microsoft.com/azure/quantum/)—Conceptual documentation and a full reference to everything about Q#, including changes and additions since this book was printed
- *Quantum Development Kit samples* (https://github.com/microsoft/quantum)—Complete samples for using Q#, both on its own and with host programs in Python and .NET, covering a wide range of different applications
- *QuTiP.org* (http://qutip.org)—Full user's guide for the QuTiP package we used to help with the math in this book

There are also some great communities for quantum computing experts and novices alike. Joining a quantum development community like the following can help resolve questions you have along the way and will also let you assist others with their journeys:

- *qsharp.community* (https://qsharp.community)—A community of Q# users and developers, complete with chat room, blog, and project repositories
- *Quantum Computing Stack Exchange* (https://quantumcomputing.stackexchange.com/)—A great place to ask for answers to quantum computing questions, including any Q# questions you may have
- *Women in Quantum Computing and Applications* (https://wiqca.dev)—An inclusive community for people of all genders to celebrate quantum computing and the people who make it possible
- *Quantum Open Source Foundation* (https://qosf.org/)—A community supporting the development and standardization of open tools for quantum computing

Copyrighted material

- *Unitary Fund* (https://unitary.fund/)—A nonprofit working to create a quantum technology ecosystem that benefits the most people

### Going further

Quantum computing is a fascinating new field that offers new ways of thinking about computation and new tools for solving difficult problems. This book can help you get a start in quantum computing so that you can continue to explore and learn. That said, this book isn't a textbook and isn't intended to prepare you for quantum computing research all on its own. As with classical algorithms, developing new quantum algorithms is a mathematical art as much as anything else; while we touch on math in this book and use it to explain algorithms, a variety of textbooks are available that can help you build on the ideas we cover.

Once you've read this book and gotten started with quantum computing, if you want to continue your journey into physics or mathematics, we suggest one of the following resources:

- The Complexity Zoo (https://complexityzoo.net/Complexity_Zoo)
- The Quantum Algorithm Zoo (http://quantumalgorithmzoo.org)
- *Complexity Theory: A Modern Approach* by Sanjeev Arora and Boaz Barak (Cambridge University Press, 2009)
- *Quantum Computing: A Gentle Introduction* by Eleanor G. Rieffel and Wolfgang H. Polak (MIT Press, 2011)
- *Quantum Computing since Democritus* by Scott Aaronson (Cambridge University Press, 2013)
- *Quantum Computation and Quantum Information* by Michael A. Nielsen and Isaac L. Chuang (Cambridge University Press, 2000)
- *Quantum Processes Systems, and Information* by Benjamin Schumacher and Michael Westmoreland (Cambridge University Press, 2010)

Copyrighted material

# *about the authors*

**SARAH KAISER** completed her PhD in physics (quantum information) at the University of Waterloo's Institute for Quantum Computing. She has spent much of her career developing new quantum hardware in the lab, from building satellites to hacking quantum cryptography hardware. Communicating what is so exciting about quantum is her passion, and she loves building new demos and tools to help enable the quantum community to grow. When not at her mechanical keyboard, she loves kayaking and writing books about science for all ages.

**CHRIS GRANADE** completed their PhD in physics (quantum information) at the University of Waterloo's Institute for Quantum Computing and now works in the Quantum Systems group at Microsoft. They work in developing the standard libraries for Q# and are an expert in the statistical characterization of quantum devices from classical data. Chris also helped Scott Aaronson prepare his lectures as a book, *Quantum Computing Since Democritus* (Cambridge University Press, 2013).



Copyrighted material

# *about the cover illustration*

The figure on the cover of *Learn Quantum Computing with Python and Q#* is captioned "Hongroise," or Hungarian woman. The illustration is taken from a collection of dress costumes from various countries by Jacques Grasset de Saint-Sauveur (1757–1810), titled Costumes de Différents Pays, published in France in 1797. Each illustration is finely drawn and colored by hand. The rich variety of Grasset de Saint-Sauveur's collection reminds us vividly of how culturally apart the world's towns and regions were just 200 years ago.

Isolated from each other, people spoke different dialects and languages. In the streets or in the countryside, it was easy to identify where they lived and what their trade or station in life was just by their dress. The way we dress has changed since then and the diversity by region, so rich at the time, has faded away. Itis now hard to tell apart the inhabitants of different continents, let alone different towns, regions, or countries. Perhaps we have traded cultural diversity for a more varied personal life—certainly for a more varied and fast-paced technological life.

At a time when it is hard to tell one computer book from another, Manning celebrates the inventiveness and initiative of the computer business with book covers based on the rich diversity of regional life of two centuries ago, brought back to life by Grasset de Saint-Sauveur's pictures.



Copyrighted material

# 1 Introducing quantum computing

**_This chapter covers_**

- Why people are excited about quantum computing
- What a quantum computer is
- What a quantum computer can and cannot do
- How quantum computers relate to classical programming

Quantum computing has been an increasingly popular research field and source of hype over the last few years. By using quantum physics to perform computation in new and wonderful ways, *quantum computers* can impact society, making it an exciting time to get involved and learn how to program quantum computers and apply quantum resources to solve problems that matter.

In all the buzz about the advantages quantum computing offers, however, it is easy to lose sight of the *real* scope of those benefits. We have some interesting historical precedent for what can happen when promises about a technology outpace reality. In the 1970s, machine learning and artificial intelligence suffered from dramatically reduced funding, as the hype and excitement around AI outstripped its results; this would later be called the "AI winter." Similarly, internet companies faced the same danger when trying to overcome the dot-com bust.



Copyrighted material

One way forward is to critically understand the promise offered by quantum computing, how quantum computers work, and what is and is not in scope for quantum computing. In this chapter, we help you develop that understanding so that you can get hands-on and write your own quantum programs in the rest of the book.

All that aside, though, it's just really cool to learn about an entirely new computing model! As you read this book, you'll learn how quantum computers work by programming simulations that you can run on your laptop today. These simulations will show many essential elements of what we expect real commercial quantum programming to be like while useful commercial hardware is coming online. This book is intended for folks who have some basic programming and linear algebra experience but no prior knowledge about quantum physics or computing. If you have some quantum familiarity, you can jump into parts 2 and 3, where we get into quantum programming and algorithms.

## 1.1 Why does quantum computing matter?

Computing technology is advancing at a truly stunning pace. Three decades ago, the 80486 processor allowed users to execute 50 MIPS (million instructions per second). Today, small computers like the Raspberry Pi can reach 5,000 MIPS, while desktop processors can easily reach 50,000 to 300,000 MIPS. If we have an exceptionally difficult computational problem we'd like to solve, a very reasonable strategy is to simply wait for the next generation of processors to make our lives easier, our videos stream faster, and our games more colorful.

For many problems that we care about, however, we're not so lucky. We might hope that getting a CPU that's twice as fast will let us solve problems that are twice as big, but as with so much in life, "more is different." Suppose we sort a list of 10 million numbers and find that it takes about 1 second. Later, if we want to sort a list of 1 billion numbers in 1 second, we'll need a CPU that's 130 times faster, not just 100 times. When solving some kinds of problems, this gets even worse: for some graphics problems, going from 10 million to 1 billion points would take 13,000 times longer.

Problems as widely varied as routing traffic in a city and predicting chemical reactions become more difficult *much* more quickly. If quantum computing was about making a computer that runs 1,000 times as fast, we would barely make a dent in the daunting challenges that we want to solve. Fortunately, quantum computers are *much* more interesting. We expect that quantum computers will be much *slower* than classical computers but that the resources required to solve many problems will *scale* differently, such that if we look at the right kinds of problems, we can break through "more is different." At the same time, quantum computers aren't a magic bullet—some problems will remain hard. For example, while it is likely that quantum computers can help us immensely with predicting chemical reactions, they may not be much help with other difficult problems.

Investigating exactly which problems we can obtain such an advantage in and developing quantum algorithms to do so has been a large focus of quantum computing

Copyrighted material

research. Up until this point, it has been very difficult to assess quantum approaches this way, as doing so required extensive mathematical skill to write out quantum algorithms and understand all the subtleties of quantum mechanics.

As industry has started developing platforms to help connect developers to quantum computing, however, this situation has begun to change. By using Microsoft's entire Quantum Development Kit, we can abstract away most of the mathematical complexities of quantum computing and begin actually *understanding* and *using* quantum computers. The tools and techniques taught in this book allow developers to explore and understand what writing programs for this new hardware platform will be like.

Put differently, quantum computing is not going away, so understanding what problems we can solve with it matters quite a lot indeed! Independent of whether a quantum "revolution" happens, quantum computing has factored—and will continue to factor—heavily into decisions about how to develop computing resources over the next several decades. Decisions like these are strongly impacted by quantum computing:

- What assumptions are reasonable in information security?
- What skills are useful in degree programs?
- How can we evaluate the market for computing solutions?

For those of us working in tech or related fields, we increasingly must make such decisions or provide input for them. We have a responsibility to understand what quantum computing is and, perhaps more important, what it is not. That way, we will be best prepared to step up and contribute to these new efforts and decisions.

All that aside, another reason quantum computing is such a fascinating topic is that it is both similar to and very different from classical computing. Understanding both the similarities and differences between classical and quantum computing helps us understand what is fundamental about computing in general. Both classical and quantum computation arise from different descriptions of physical laws such that understanding computation can help us understand the universe in a new way.

What's absolutely critical, though, is that there is no one right or even best reason to be interested in quantum computing. Whatever brings you to quantum computing research or applications, you'll learn something interesting along the way.

## 1.2 What is a quantum computer?

Let's talk a bit about what actually makes up a quantum computer. To facilitate this discussion, let's briefly talk about what the term *computer* means.

> **DEFINITION** A *computer* is a device that takes data as input and does some sort of operations on that data.

There are many examples of what we have called a *computer*; see figure 1.1 for some examples.

All of these have in common that we can model them with classical physics—that is, in terms of Newton's laws of motion, Newtonian gravity, and electromagnetism.

Copyrighted material

Copyrighted image

Figure 1.1 Several examples of different kinds of computers, including the UNIVAC mainframe operated by Rear Admiral Hopper, a room of "human computers" working to solve flight calculations, a mechanical calculator, and a LEGO-based Turing machine. Each computer can be described by the same mathematical model as computers like cell phones, laptops, and servers. Sources: Photo of "human computers" by NASA. Photo of LEGO Turning machine by Projet Rubens, used under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/).

This will help us tell apart the kinds of computers we're used to (e.g., laptops, phones, bread machines, houses, cars, and pacemakers) and the computers we're learning about in this book. To tell the two apart, we'll call computers that can be described using classical physics *classical computers*. What's nice about this is that if we replace the term *classical physics* with *quantum physics*, we have a great definition for what a quantum computer is!

> **DEFINITION** A *quantum computer* is a device that takes data as input and does some sort of operations on that data with a process that can only be described using quantum physics.

Put differently, the distinction between classical and quantum computers is precisely that between classical and quantum physics. We will get into this more later in the book. But the primary difference is one of scale: our everyday experience is largely with objects that are large enough and hot enough that even though quantum effects still exist, they don't do much on average. While quantum mechanics works even at the scale of everyday objects like coffee mugs, bags of flour, and baseball bats, it turns out that we can do a very good job of describing how these objects interact using classical physics alone.

Copyrighted material

**Deep dive: What happened to relativity?**

Quantum physics applies to objects that are very small and very cold or well isolated. Similarly, another branch of physics called *relativity* describes objects that are large enough for gravity to play an important role or that are moving very fast—near the speed of light. Many computers rely on relativistic effects; indeed, global positioning satellites depend critically on relativity. So far, we have primarily been comparing classical and quantum physics, so what about relativity?

As it turns out, all computation that is implemented using relativistic effects can also be described using purely classical models of computing such as Turing machines. By contrast, quantum computation cannot be described as faster classical computation but requires a different mathematical model. There has not yet been a proposal for a "gravitic computer" that uses relativity in the same way, so we're safe to set relativity aside in this book.

If we focus on a much smaller scale where quantum mechanics *is* needed to describe our systems, then quantum computing is the art of using small, well-isolated devices to usefully transform data in ways that cannot be described in terms of classical physics alone. One way to build quantum devices is to use small classical computers such as digital signal processors (DSPs) to control the properties of exotic materials.

Copyrighted image

Quantum devices may differ in the details of how they are controlled, but ultimately all quantum devices are controlled from and read out by classical computers and control electronics of some kind. After all, we are interested in classical data, so there must eventually be an interface with the classical world.

**NOTE** Most quantum devices must be kept very cold and well isolated, since they can be extremely susceptible to noise.

By applying quantum operations using embedded classical hardware, we can manipulate and transform quantum data. The power of quantum computing then comes from carefully choosing which operations to apply in order to implement a useful transformation that solves a problem of interest.

Copyrighted material

- Massive GPU clusters
- Reprogrammable hardware (e.g., Catapult/Brainwave)
- Tensor processing unit (TPU) clusters
- High-permanence, high-latency archival storage (e.g., Amazon Glacier)

Going forward, cloud services like Azure Quantum (https://azure.com/quantum) will make the power of quantum computing available in much the same way.

Just as high-speed, high-availability internet connections have made cloud computing accessible for large numbers of users, we will be able to use quantum computers from the comfort of our favorite WiFi-blanketed beach or coffee shop or even from a train as we view majestic mountain ranges in the distance.

### 1.3.1 What can quantum computers do?

As quantum programmers, if we have a particular problem, *how do we know it makes sense to solve it with a quantum computer*?

We are still learning about the exact extent of what quantum computers are capable of, and thus we don't have any concrete rules to answer this question yet. So far, we have found some examples of problems where quantum computers offer significant advantages over the best-known classical approaches. In each case, the quantum algorithms that have been found to solve these problems exploit quantum effects to achieve the advantages, sometimes referred to as a *quantum advantage*. The following are two useful quantum algorithms:

- Grover's algorithm (discussed in chapter 11) searches a list of $N$ items in $\sqrt{N}$ steps.
- Shor's algorithm (chapter 12) quickly factors large integers, such as those used by cryptography to protect private data.

We'll see several more in this book, but Grover's and Shor's are good examples of how quantum algorithms work: each uses quantum effects to separate correct answers to computational problems from invalid solutions. One way to realize a quantum advantage is to find ways of using quantum effects to separate correct and incorrect solutions to classical problems.

Copyrighted image

Copyrighted material

Copyrighted image

Quantum computers also offer significant benefits for simulating properties of quantum systems, opening up applications to quantum chemistry and materials science. For instance, quantum computers could make it much easier to learn about the ground-state energies of chemical systems. These ground-state energies then provide insight into reaction rates, electronic configurations, thermodynamic properties, and other properties of immense interest in chemistry.

Along the way to developing these applications, we have also seen significant advantages in spin-off technologies such as quantum key distribution and quantum metrology, some of which we will see in the next few chapters. In learning to control and understand quantum devices for the purpose of computing, we have also learned valuable techniques for imaging, parameter estimation, security, and more. While these are not applications for quantum computing in a strict sense, they go a long way toward showing the values of *thinking* in terms of quantum computation.

Of course, new applications of quantum computers are much easier to discover when we have a concrete understanding of how quantum algorithms work and how to build new algorithms from basic principles. From that perspective, quantum programming is a great resource to learn how to discover entirely new applications.

### 1.3.2 What can't quantum computers do?

Like other forms of specialized computing hardware, quantum computers won't be good at everything. For some problems, classical computers will simply be better suited to the task. In developing applications for quantum devices, it's helpful to note what tasks or problems are out of scope for quantum computing.

The short version is that we don't have any hard-and-fast rules to quickly decide which tasks are best run on classical computers and which tasks can take advantage of quantum computers. For example, the storage and bandwidth requirements for Big Data–style applications are very difficult to map onto quantum devices, where we may only have a relatively small quantum system. Current quantum computers can only record inputs of no more than a few dozen bits, and this limitation will become more relevant as quantum devices are used for more demanding tasks. Although we expect to eventually build much larger quantum systems than we can now, classical computers will likely always be preferable for problems that require large amounts of input/output to solve.

Similarly, machine learning applications that depend heavily on random access to large sets of classical inputs are conceptually difficult to solve with quantum computing. That said, there *may* be other machine learning applications that map much more

Copyrighted material

naturally onto quantum computation. Research efforts to find the best ways to apply quantum resources to solve machine learning tasks are still ongoing. In general, problems that have small input and output data sizes but require large amounts of computation to get from input to output are good candidates for quantum computers.

In light of these challenges, it might be tempting to conclude that quantum computers *always* excel at tasks that have small inputs and outputs but very intense computation between the two. Notions like *quantum parallelism* are popular in media, and quantum computers are sometimes even described as using parallel universes to compute.

> **NOTE** The concept of "parallel universes" is a great example of an analogy that can help make quantum concepts understandable but can lead to nonsense when taken to its extreme. It can sometimes be helpful to think of the different parts of a quantum computation as being in different universes that can't affect each other, but this description makes it harder to think about some of the effects we will learn in this book, such as interference. When taken too far, the parallel-universes analogy also lends itself to thinking of quantum computing in ways that are closer to a particularly pulpy and fun episode of a sci-fi show like *Star Trek* than to reality.

What this fails to communicate, however, is that it isn't always obvious how to use quantum effects to extract useful answers from a quantum device, even if the state of the quantum device appears to contain the desired output. For instance, one way to factor an integer $N$ using a classical computer is to list each *potential* factor and check whether it's actually a factor or not:

1. Let $i = 2$.
2. Check if the remainder of $N / i$ is zero.
   - If so, return that $i$ factors $N$.
   - If not, increment $i$ and loop.

We can speed up this classical algorithm by using a large number of different classical computers, one for each potential factor that we want to try. That is, this problem can be easily parallelized. A quantum computer can try each potential factor within the same device, but as it turns out, this isn't *yet* enough to factor integers faster than the classical approach. If we use this approach on a quantum computer, the output will be one of the potential factors chosen at random. The actual correct factors will occur with probability about $1 / \sqrt{N}$, which is no better than the classical algorithm.

As we'll see in chapter 12, though, we can use other quantum effects to factor integers with a quantum computer faster than the best-known classical factoring algorithms. Much of the heavy lifting done by Shor's algorithm is to make sure that the probability of measuring a correct factor at the end is much larger than the probability of measuring an incorrect factor. Canceling out incorrect answers this way is where much of the art of quantum programming comes in; it's not easy or even possible to do for all problems we might want to solve.

Copyrighted material

To understand more concretely what quantum computers can and can't do and how to do cool things with quantum computers despite these challenges, it's helpful to take a more concrete approach. Thus, let's consider what a quantum program even **is**, so that we can start writing our own.

## 1.4 What is a program?

Throughout this book, we will often find it useful to explain a quantum concept by first reexamining the analogous classical concept. In particular, let's take a step back and examine what a classical program is.

> **DEFINITION** A *program* is a sequence of instructions that can be interpreted by a classical computer to perform a desired task. Tax forms, driving directions, recipes, and Python scripts are all examples of programs.

We can write classical programs to break down a wide variety of different tasks for interpretation by all sorts of different computers. See figure 1.4 for some example programs.

Copyrighted image



**Figure 1.4 Examples of classical programs. Tax forms, map directions, and recipes are all examples in which a sequence of instructions is interpreted by a classical computer such as a person. These may look very different, but each uses a list of steps to communicate a procedure.**

Copyrighted material

Let's take a look at what a simple "hello, world" program might look like in Python:

```
>>> def hello():
...     print("Hello, world!")
...
>>> hello()
Hello, world!
```

At its most basic, this program can be thought of as a sequence of instructions given to the Python *interpreter*, which then executes each instruction in turn to accomplish some effect—in this case, printing a message to the screen. That is, the program is a *description* of a task that is then *interpreted* by Python and, in turn, by our CPU to accomplish our goal. This interplay between description and interpretation motivates calling Python, C, and other such programming tools *languages*, emphasizing that programming is how we communicate with our computers.

In the example of using Python to print "Hello, world!" we are effectively communicating with Guido van Rossum, the founding designer of the Python language. Guido then effectively communicates on our behalf with the designers of the operating system we are using. These designers in turn communicate on our behalf with Intel, AMD, ARM, or whatever company designed the CPU we are using, and so forth.

### 1.4.1 What is a quantum program?

Like classical programs, quantum programs consist of sequences of instructions that are interpreted by classical computers to perform a particular task. The difference, however, is that in a quantum program, the task we wish to accomplish involves controlling a quantum system to perform a computation.

As a result, the instructions used in classical and quantum programs differ as well. A classical program may describe a task such as loading some cat pictures from the internet in terms of instructions to a networking stack and eventually in terms of assembly instructions such as `mov` (move). By contrast, quantum languages like Q# allow programmers to express quantum tasks in terms of instructions like `M` (measure). When run using quantum hardware, these programs may instruct a digital signal processor to send microwaves, radio waves, or lasers into a quantum device and amplify signals coming out of the device.

Throughout the rest of this book, we will see many examples of the kinds of tasks a quantum program is faced with solving, or at least addressing, and what kinds of classical tools we can use to make quantum programming easier. For example, figure 1.5 shows an example of writing a quantum program in Visual Studio Code, a classical integrated development environment (IDE).

Copyrighted material

Copyrighted image

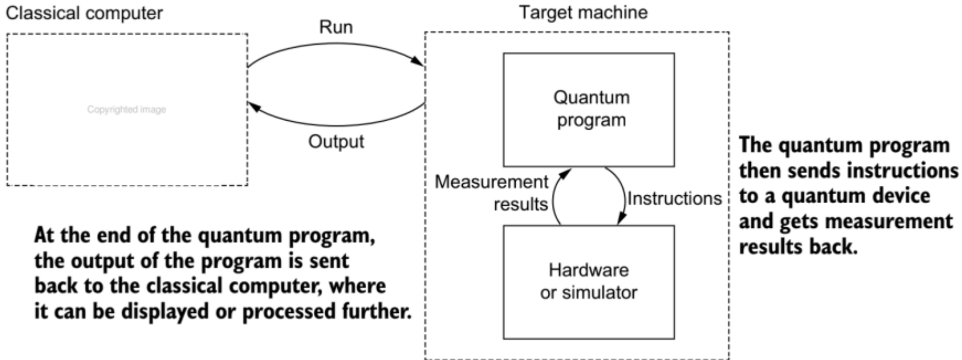
***This chapter covers***

- Why random numbers are an important resource
- What is a qubit?
- What basic operations can we perform on a qubit?
- Programming a quantum random number generator in Python

In this chapter, we'll start to get our feet wet with some quantum programming concepts. The main concept we will explore is the *qubit*, the quantum analogue of a classical bit. We'll use qubits as an abstraction or model to describe the new kinds of computing that are possible with quantum physics. Figure 2.1 shows a model of using a quantum computer as well as the simulator setup that we use in this book. Real or simulated qubits will live on the target machine and interact with the quantum programs that we will be writing! Those quantum programs can be sent by various host programs that then wait to receive the results from the quantum program.



Copyrighted material



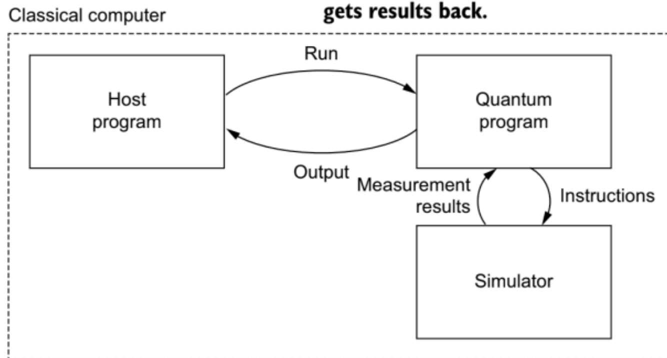




**Figure 2.1 A mental model for how we can use a quantum computer. The top half of the figure is the general model for a quantum computer. We will be using local simulators for this book, and the bottom half represents what we will be building and using.**

To help learn about what qubits are and how we interact with them, we will use an example of how they are used today: random number generation. While we can build up much more interesting devices from qubits, the simple example of a *quantum random number generator* (QRNG) is a good way to get familiar with the qubit.

Copyrighted material

## 2.1 Why do we need random numbers?

Humans like certainty. We like it when pressing a key on our keyboard does the same thing every time. However, there are some contexts in which we want randomness:

- Playing games
- Simulating complex systems (such as the stock market)
- Picking secure secrets (for example, passwords and cryptographic keys)

In all of these situations where we want randomness, we can describe the chances of each outcome. For random events, describing the chances is all we can say about the situation until the die is cast (or the coin is flipped or the password is reused). When we describe the chances of each example, we say things like this:

- *If* I roll this die, *then* I will get a six *with probability* 1 out of 6.
- *If* I flip this coin, *then* I will get heads *with probability* 1 out of 2.

We can also describe cases where the probabilities aren't the same for every outcome. On *Wheel of Fortune*, (figure 2.2), the probability that *if* we spin the wheel, *then* we will get a $1,000,000 prize is much smaller than the probability that *if* we spin the wheel, *then* we will go bankrupt.

Copyrighted image

**Figure 2.2 Probabilities of $1,000,000 and Bankrupt on *Wheel of Fortune*. Before spinning the wheel, we don't know exactly where it will land, but we do know by looking at the wheel that the probability of getting Bankrupt is much larger than the probability of winning big.**

As on game shows, there are many contexts in computer science where randomness is critical, especially when security is required. If we want to keep some information private, cryptography lets us do so by combining our data with random numbers in different ways. If our random number generator isn't very good—that is to say, if an attacker can predict what numbers we use to protect our private data—then cryptography doesn't

Copyrighted material

help us much. We can also imagine using a poor random number generator to run a raffle or a lottery; an attacker who figures out how our random numbers are generated can take us straight to the bank.

Copyrighted image

As it turns out, quantum mechanics lets us build some really unique sources of randomness. If we build them right, the randomness of our results is guaranteed by *physics*, not an assumption about how long it would take for a computer to solve a difficult problem. This means a hacker or adversary would have to break the laws of physics to break the security! This does not mean we should use quantum random numbers for everything; humans are still the weakest link in security infrastructure 😼.

**Deep dive: Computational security and information theoretic security**

Some ways of protecting private information rely on assumptions about what problems are easy or hard for an attacker to solve. For instance, the RSA algorithm is a commonly used encryption algorithm and is based on the difficulty of finding prime factors for large numbers. RSA is used on the web and in other contexts to protect user data, under the assumption that adversaries can't easily factor very large numbers. So far, this has proven to be a good assumption, but it is entirely possible that a new factoring algorithm may be discovered, undermining the security of RSA. New models of computation like quantum computing also change how reasonable or unreasonable it is to make computational assumptions like "factoring is hard." As we'll see in chapter 11, a quantum algorithm known as *Shor's algorithm* allows for solving some kinds of cryptographic problems much faster than classical computers can, challenging the assumptions that are commonly used to promise computational security.

By contrast, if an adversary can only ever randomly guess at secrets, even with very large amounts of computing power, then a security system provides much better guarantees about its ability to protect private information. Such systems are said to be *informationally secure*. Later in this chapter, we'll see that generating random numbers in a hard-to-predict fashion allows us to implement an informationally secure procedure called a *one-time pad*.

This gives us some confidence that we can use quantum random numbers for vital tasks, such as to protect private data, run lotteries, and play *Dungeons and Dragons*. Simulating how quantum random number generators work lets us learn many of the basic concepts underlying quantum mechanics, so let's jump right in and get started!

Copyrighted material

As mentioned earlier, one great way to get started is to look at an example of a quantum program that generates random numbers: a quantum random number generator (QRNG). Don't worry if the following algorithm (also shown in figure 2.3) doesn't make a lot of sense right now—we'll explain the different pieces as we go through the rest of the chapter:

1. Ask the quantum device to allocate a qubit.
2. Apply an instruction called the *Hadamard instruction* to the qubit; we learn about this later in the chapter.
3. Measure the qubit, and return the result.

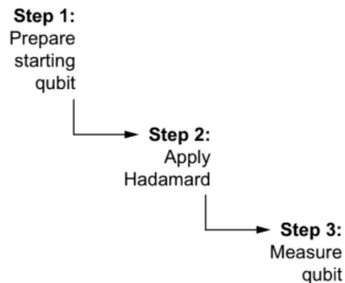



**Figure 2.3 Quantum random number generator algorithm. To sample random numbers with a quantum computer, our program will prepare a fresh qubit and then use a Hadamard instruction to prepare the superposition we need. Finally, we can measure and return the random result that we get at the end.**

In the rest of the chapter, we'll develop a Python class called `QuantumDevice` to let us write programs that implement algorithms like this one. Once we have a `QuantumDevice` class, we'll be able to write QRNG as a Python program similar to classical programs that we're used to.

**NOTE** Please see appendix A for instructions on how to set up Python on your device to run quantum programs.

Note that the following sample will not run until you have written the simulator in this chapter 😊.

**Listing 2.1 qrng.py: a quantum program that generates random numbers**

```
def qrng(device : QuantumDevice) -> bool:
    with device.using_qubit() as q:
        q.h()
        return q.measure()
```

**Quantum programs are written just like classical programs. In this case, we're using Python, so our quantum program is a Python function qrng that implements a QRNG.**

**Quantum programs work by asking quantum computing hardware for qubits: quantum analogues of bits that we can use to perform computations.**

**Once we have a qubit, we can issue instructions to that qubit. Similar to assembly languages, these instructions are often denoted by short abbreviations; we'll see what h() stands for later in this chapter.**

**To get data back from our qubits, we can measure them. In this case, half of the time, our measurement will return True, and the other half of the time, we'll get back False.**

Copyrighted material

into 1 bits and vice versa. In classical storage devices like hard drives, a NOT gate flips the magnetic field that stores our bit value. As shown in figure 2.5, we can think of NOT as implementing a 180° rotation between the 0 and 1 points we drew in figure 2.4.

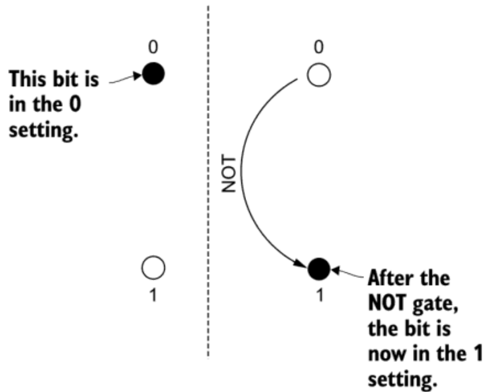



**Figure 2.5 The classical NOT operation flips a classical bit between 0 and 1. For instance, if a bit starts in the 0 state, NOT flips it to the 1 state.**

Visualizing classical bits this way also lets us extend the notion of bits slightly to include a way to describe *random* bits (which will be helpful later). If we have a *fair coin* (that is, a coin that lands heads half the time and tails the other half), then it wouldn't be correct to call that coin a 0 or a 1. We only know what bit value our coin bit has if we set it with a particular side face up on a surface; we can also flip it to get a random bit value. Every time we flip a coin, we know that eventually, it will land, and we will get heads or tails. Whether it lands heads or tails is governed by a probability called the *bias* of the coin. We have to pick a side of the coin to describe the bias, which is easy to phrase as a question like "What is the probability that the coin will land heads?" Thus a fair coin has a bias of 50% because it lands with the value heads half of the time, which is mapped to the bit value 0 in figure 2.6.

Using this visualization, we can take our previous two dots indicating the bit values 0 and 1 and connect them with a line on which we can plot our coin's bias. It becomes

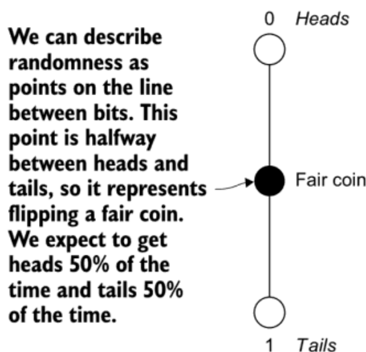



**Figure 2.6 We can use the same picture as before to extend our concept of a bit to describe a coin. Unlike a bit, a coin has a probability of being either 0 or 1 each time it is tossed. We graphically represent that probability as a point between 0 and 1.**

Copyrighted material

easier to see that a NOT operation (which still works on our new probabilistic bit) doesn't do anything to a fair coin. If 0 and 1 occur with the same probability, then it doesn't matter if we rotate a 0 to a 1 or a 1 to a 0: we'll still wind up with 0 and 1 having the same probability.

What if our bias is not in the middle? If we know that someone is trying to cheat by using a weighted or modified coin that almost always lands on heads, we can say the bias of the coin is 90% and plot it on our line by drawing a point much closer to 0 than to 1.

> **DEFINITION** The point on a line where we would draw each classical bit is the *state* of that bit.

Let's consider a scenario. Say I want to send you a bunch of bits stored using padlocks. What is the cheapest way I can do so?

One approach is to mail a box containing many padlocks that are either open or closed and hope that they arrive in the same state in which I sent them. On the other hand, we can agree that all padlocks start in the 0 (unlocked) state, and I can send you instructions on which padlocks to close. This way, you can buy your own padlocks, and I only need to send a *description* of how to prepare those padlocks using classical NOT gates. Sending a piece of paper or an email is much cheaper than mailing a box of padlocks!

This illustrates a principle we will rely on throughout the book: *the state of a physical system can also be described in terms of instructions for how to prepare that state.* Thus, the operations allowed on a physical system also define what states are possible.

Although it may sound completely trivial, there is one more thing we can do with classical bits that will turn out to be critical for how we understand quantum computing: we can look at them. If I look at a padlock and conclude, "Aha! That padlock is unlocked," then I can now think of my brain as a particularly squishy kind of bit. The 0 message is stored in my brain by my thinking, "Aha! That padlock is unlocked," while a 1 message would be stored by my thinking, "Ah, well, that padlock is locked ☹." In effect, by looking at a classical bit, I have *copied* it into my brain. We say that the act of *measuring* the classical bit copies that bit.

More generally, modern life is built around the ease with which we copy classical bits by looking at them. We copy classical bits with truly reckless abandon, measuring many billions of classical bits every second that we copy data from our video game consoles to our TVs.

On the other hand, if a bit is stored as a coin, then the process of measuring involves flipping it. Measuring doesn't quite copy the coin, as I might get a different measurement result the next time I flip. If I only have one measurement of a coin, I can't conclude the probability of getting heads or tails. We didn't have this ambiguity with padlock bits because we knew the state of the padlocks was either 0 or 1. If I measured a padlock and found it to be in the 0 state, I would know that it would always be in the 0 state unless I did something to the padlock.

Copyrighted material

The situation isn't precisely the same in quantum computing, as we'll see later in the chapter. While measuring classical information is cheap enough that we complain about precisely how many billions of bits a $5 cable lets us measure, we have to be much more careful with how we approach quantum measurements.

### 2.2.2 Abstractions are our friend

Regardless of how we physically build a bit, we can (fortunately) represent them the same way in both math and code. For instance, Python provides the `bool` type (short for Boolean, in honor of the logician George Boole), which has two valid values: `True` and `False`. We can represent transformations on bits such as NOT and OR as operations acting on `bool` variables. Importantly, we can specify a classical operation by describing how that operation transforms each possible input, often called a *truth table*.

**DEFINITION** A *truth table* is a table describing the output of a classical operation for every possible combination of inputs. For example, figure 2.7 shows the truth table for the AND operation.

**Truth tables are one way to show what happens to classical bits in functions or logical circuits.**

| Input | Output |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

**same output bits as the truth table to the left.**

**Figure 2.7 Truth table for the logical operation AND. If we know the entire truth table for a logical operation, then we know what that operation does for any possible input.**

We can find the truth table for the NAND (short for NOT-AND) operation in Python by iterating over combinations of `True` and `False`.

**Listing 2.2 Using Python to print out a truth table for NAND**

```
>>> from itertools import product
>>> for inputs in product([False, True], repeat=2):
...     output = not (inputs[0] and inputs[1])
...     print(f"{inputs[0]}\t{inputs[1]}\t->\t{output}")
False   False   ->      True
False   True    ->      True
True    False   ->      True
True    True    ->      False
```

**NOTE** Describing an operation as a truth table holds for more complicated operations. In principle, even an operation like addition between two 64-bit

Copyrighted material

integers can be written as a truth table. This isn't very practical, though, as a truth table for two 64-bit inputs would have $2^{128} \approx \times 10^{38}$ entries and would take $10^{40}$ bits to write. By comparison, recent estimates put the size of the entire internet at closer to $10^{27}$ bits.

Much of the art of classical logic and hardware design is making *circuits* that can provide very compact representations of classical operations rather than relying on potentially massive truth tables. In quantum computing, we use the name *unitary operators* for similar truth tables for quantum bits, which we will expand on as we go along.

In summary:

- Classical bits are physical systems that can be in one of two different *states*.
- Classical bits can be manipulated through *operations* to process information.
- The act of *measuring* a classical bit makes a copy of the information contained in the state.

**NOTE** In the next section, we'll use linear algebra to learn about *qubits*, the basic unit of information in a quantum computer. If you need a refresher on linear algebra, this would be a great time to take a detour to appendix C. We'll refer to an analogy from this appendix throughout the book, where we'll think of vectors as directions on a map. We'll be right here when you get back!

## 2.3 Qubits: States and operations

Just as classical bits are the most basic unit of information in a classical computer, *qubits* are the basic unit of information in a quantum computer. They can be physically implemented by systems that have two states, just like classical bits, but they behave according to the laws of quantum mechanics, which allows for some behaviors that classical bits are not capable of. Let's treat qubits like we would any other fun new computer part: plug it in and see what happens!

Copyrighted image

Copyrighted material

Copyrighted image

### 2.3.1 State of the qubit

To implement our QRNG, we need to work out how to describe our qubit. We have used locks, baseballs, and other classical systems to represent *classical* bit values of 0 or 1. We can use many physical systems to act as our qubit, and *states* are the "values" our qubit can have.

Similar to the 0 and 1 states of classical bits, we can write labels for quantum states. The qubit states that are most similar to the classical 0 and 1 are $|0\rangle$ and $|1\rangle$, as shown in figure 2.8. These are referred to as *ket 0* and *ket 1,* respectively.

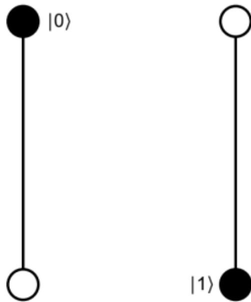



**Figure 2.8 Using Dirac (bra-ket) notation for qubits, we can graphically represent the $|0\rangle$ and $|1\rangle$ states for qubits the same way as we represented the 0 and 1 states of classical bits. In particular, we'll draw the $|0\rangle$ and $|1\rangle$ states as opposite points along an axis, as shown here.**

Copyrighted image

One thing to be mindful of, though, is that a state is a convenient model used to predict how a qubit behaves, not an inherent property of the qubit. This distinction becomes especially important when we consider measurement later in the chapter—as we will see, we cannot directly measure the state of a qubit.

Copyrighted material

If we rotate a qubit in the $|0\rangle$ state clockwise by 90° instead of 180°, we get a quantum operation that we can think of as the square root of a NOT operation, as shown in figure 2.10.

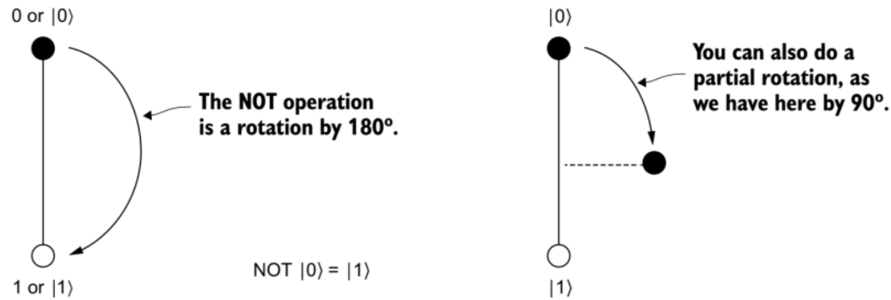



**Figure 2.10 We can also rotate a state by less than 180 degrees. Doing so, we get a state that is neither |0⟩ nor |1⟩ but that lies halfway around the circle between them.**

Just as we earlier defined the square root $\sqrt{x}$ of a number $x$ as being a number $y$ such that $y^2 = x$, we can define the square root of a quantum operation. If we apply a 90° rotation twice, we get back the NOT operation, so we can think of the 90° rotation as the square root of NOT.

Copyrighted image

We now have a new state that is neither $|0\rangle$ nor $|1\rangle$, but an equal combination of them both. In precisely the same sense that we can describe "northeast" by adding together the directions "north" and "east," we can write this new state as shown in figure 2.11.

The state of a qubit can be represented as a point on a circle that has two labeled states on the poles: $|0\rangle$ and $|1\rangle$. More generally, we will picture rotations using arbitrary angles $\theta$ between qubit states, as shown in figure 2.12.

Copyrighted material

**Just as we can point ourselves northeast by looking north and then rotating toward the east, we get a new state that points between $|0\rangle$ and $|1\rangle$ by starting in $|0\rangle$ and rotating toward $|1\rangle$.**

**In the same way we might think of directions like north and east, quantum states like $|0\rangle$ and $|1\rangle$ are directions.**

$$\cos(90^\circ / 2)\ |0\rangle + \sin(90^\circ / 2)\ |1\rangle = (|0\rangle + |1\rangle) / \sqrt{2}$$

**We can rotate between states using the same math we use to describe rotations of map directions; we just have to watch out for factors of 2.**

**For example, if we want to rotate the $|0\rangle$ state by 90°, we use the cosine and sine functions to find the new state.**

**Figure 2.11** We can write the state we get when we rotate by 90° by thinking of the $|0\rangle$ and $|1\rangle$ states as *directions*. Doing so, and using some trigonometry, we get that rotating the $|0\rangle$ state by 90° gives us a new state, $(|0\rangle + |1\rangle) / \sqrt{2}$. For more details on how to write out the math for this kind of rotation, check out appendix B for a refresher on linear algebra.

Copyrighted image

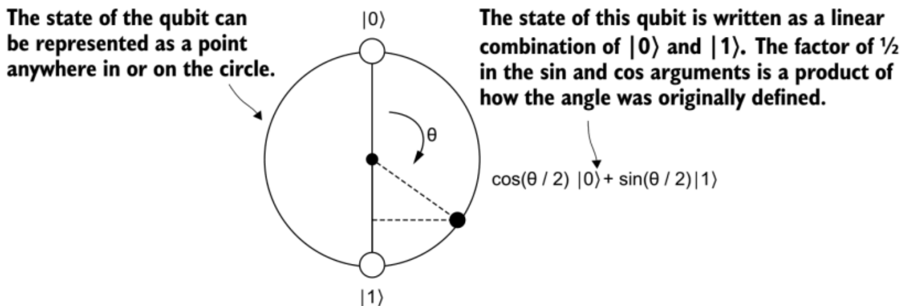



**Figure 2.12** If we rotate the $|0\rangle$ state by an angle other than 90° or 180°, the resulting state can be represented as a point on a circle that has $|0\rangle$ and $|1\rangle$ as its top and bottom poles. This gives us a way to visualize the possible states that a single qubit can be in.

Copyrighted material

Mathematically, we can write the state of any point on the circle that represents our qubit as $\cos(\theta / 2)\ |0\rangle + \sin(\theta / 2)\ |1\rangle$, where $|0\rangle$ and $|1\rangle$ are different ways of writing the vectors [[1], [0]] and [[0], [1]], respectively.

**TIP** One way to think of ket notation is that it gives *names* to vectors that we commonly use. When we write $|0\rangle$ = [[1], [0]], we're saying that [[1], [0]] is important enough that we name it after 0. Similarly, when we write $|+\rangle$ = [[1], [1]] / $\sqrt{2}$, we give a name to the vector representation of a state that we will use throughout the book.

Another way to say this is that a qubit is generally the *linear combination* of the vectors of $|0\rangle$ and $|1\rangle$ with coefficients that describe the angle that $|0\rangle$ would have to be rotated to get to the state. To be useful for programming, we can write how rotating a state affects each of the $|0\rangle$ and $|1\rangle$ states, as shown in figure 2.13.

**Let's look at rotating $|0\rangle$ by an angle θ again, and see how we can write it even when we don't know what θ is.**

**As before, we start by writing the rotation using sines and cosines.**

**Sometimes, using matrix notation instead of Dirac notation (kets) is more helpful. We can use that $|0\rangle$ = [[1], [0]] here, since both are ways of writing the same qubit state.**

$$\cos(\theta / 2)|0\rangle + \sin(\theta / 2)|1\rangle = \cos(\theta / 2)\begin{bmatrix}1\\0\end{bmatrix} + \sin(\theta / 2)\begin{bmatrix}0\\1\end{bmatrix}$$

$$= \begin{bmatrix}\cos(\theta / 2)\\ \cos(\theta / 2)\end{bmatrix}$$

**Similarly, $|1\rangle$ = [[0], [1]].**

**Once we've written each state using matrix notation, we can just add the corresponding elements together.**

**For instance, we get cos(θ / 2) for the first row, since cos(θ / 2) + 0 = cos(θ / 2).**

**Figure 2.13 Using linear algebra, we can describe the state of a single qubit as a two-element vector. In this equation, we show how that way of thinking about qubit states relates to our earlier use of Dirac (bra-ket) notation. In particular, we show the final state after rotating the $|0\rangle$ state by an arbitrary angle θ using both vector and Dirac notations; both will be helpful at different points in our quantum journey.**

**TIP** This is precisely the same as when we used a basis of vectors earlier to represent a linear function as a matrix.

We'll learn about other quantum operations in this book, but these are the easiest to visualize as rotations. Table 2.2 summarizes the states we have learned to create from these rotations.

Copyrighted material

**Table 2.2 State labels, expansions in Dirac notation, and representations as vectors**

| State label | Dirac notation | Vector representation |
|---|---|---|
| $\|0\rangle$ | $\|0\rangle$ | [[1], [0]] |
| $\|1\rangle$ | $\|1\rangle$ | [[0], [1]] |
| $\|+\rangle$ | $(\|0\rangle + \|1\rangle) / \sqrt{2}$ | [[1 / $\sqrt{2}$], [1 / $\sqrt{2}$]] |
| $\|-\rangle$ | $(\|0\rangle - \|1\rangle) / \sqrt{2}$ | [[1 / $\sqrt{2}$], [–1 / $\sqrt{2}$]] |

Copyrighted image

### 2.3.3 Measuring qubits

When we want to retrieve the information stored in a qubit, we need to measure the qubit. Ideally, we would like a measurement device that lets us directly read out all the information about the state at once. As it turns out, this is not possible by the laws of quantum mechanics, as we'll see in chapters 3 and 4. That said, measurement *can* allow us to learn information about the state relative to particular directions in the system. For instance, if we have a qubit in the $|0\rangle$ state, and we look to see if it is in the $|0\rangle$ state, we'll always get that it is. On the other hand, if we have a qubit in the $|+\rangle$ state, and we look to see if it is in the $|0\rangle$ state, we'll get a 0 outcome with 50% probability. As shown in figure 2.14, this is because the $|+\rangle$ state overlaps equally with the $|0\rangle$ and $|1\rangle$ states, such that we'll get both outcomes with the same probability.

> **TIP** Measurement outcomes of qubits are *always* classical bit values! Put differently, whether we measure a classical bit or a qubit, our result is always a classical bit.

Most of the time, we will choose to measure whether we have a $|0\rangle$ or a $|1\rangle$; that is, we'll want to measure along the line between the $|0\rangle$ and $|1\rangle$. For convenience, we give this

Copyrighted material