Student User Experience with the IBM QISKit Quantum Computing Interface

Stephan Barabasi, James Barrera, Prashant Bhalani, Preeti Dalvi, Ryan Dimiecik, Avery Leider(✉), John Mondrosch, Karl Peterson, Nimish Sawant, and Charles C. Tappert

Seidenberg School of Computer Science and Information Systems, Pace University, Pleasantville, NY 10570, USA
barabasi@us.ibm.com,
{jb05881n,pb21845n,pd21567n,rk43626p,aleider,jm58593n,
kp53279n,ns33992n,ctappert}@pace.edu

Abstract. The field of quantum computing is rapidly expanding. As manufacturers and researchers grapple with the limitations of classical silicon central processing units (CPUs), quantum computing sheds these limitations and promises a boom in computational power and efficiency. The quantum age will require many skilled engineers, mathematicians, physicists, developers, and technicians with an understanding of quantum principles and theory. There is currently a shortage of professionals with a deep knowledge of computing and physics able to meet the demands of companies developing and researching quantum technology. This study provides a brief history of quantum computing, an in-depth review of recent literature and technologies, an overview of IBMs QISKit for implementing quantum computing programs, and two successful programming examples. These two programs along with the associated Jupyter notebook pages will provide additional intermediate samples for IBM Q Experience.

Keywords: Quantum computer · Qubit · QISKit · Tutorial · Student

1 Introduction

In 1959, shortly after the arrival of the classical computer and the beginning of the digital era, theoretical physicist Richard Feynman gave a lecture series on electronic miniaturization titled “There’s Plenty of Room at the Bottom.” Feynman proposed in these lectures that the exploitation of quantum effects could create more powerful computers. Quantum computers, machines that operate on qubits rather than traditional bits, are the manifestation of Feynman’s proposal [1].

Thanks to the IBM Faculty Award that made this research possible.

The discovery of quantum mechanics in the early 20th century laid the foundations for Feynman's proposal. In 1905, Albert Einstein published a paper proposing the photon concept of light [2]. In the paper, he described light as quantum particles that are "localized points in space, which move without dividing, and which can only be produced and absorbed as complete units [2]." This challenged the accepted model at the time, which viewed light only as a wave. Ultimately, Einstein's revelation on the physical properties of light laid the groundwork for the development of quantum physics.

Following Einstein's quantum explanation of light, the field of quantum mechanics developed gradually with research by Werner Heisenberg, Niels Bohr, and Erwin Schrdinger [3]. Quantum mechanics states that the position of a particle cannot be predicted with precision as it can be in Newtonian mechanics. The only thing an observer could know about the position of a particle is the probability that it will be at a certain position at a given time [3]. By the 1980s several researchers including Feynman, Yuri Manin, and Paul Benioff had begun researching computers that operate using this concept.

The quantum bit or qubit is the basic unit of quantum information. Classical computers operate on bits using complex configurations of simple logic gates. A bit of information has two states, on or off, and is represented with a 0 or a 1. The qubit has a $|0\rangle$ state, called zero-ket and a $|1\rangle$ state called one-ket. When zero-ket is measured it produces a classical 0. When one-ket is measured it produces a classical 1. The most notable difference between classical and quantum bits is that the qubit can also be in a state that is a linear combination of both $|0\rangle$ and $|1\rangle$. This vector property of the qubit is called superposition. Superposition allows for many calculations to be performed simultaneously [4].

Another distinct property is that qubits can become entangled. Entanglement is a property of many quantum superpositions and does not have a classical analog [4]. Observation of two entangled qubits causes random behavior in the observed qubit, however, the observer can tell how the other would behave if observed in the same manner [4]. Random behavior between entangled qubits is responsible for the extra computing power of quantum machines.

Quantum computers are useful for large scale problems that classical computers lack the computational power to solve in a reasonable amount of time. Superposition creates opportunities to implement more efficient factoring, searching, sorting, modeling, and simulation algorithms. According to Almudever , factoring a 2000 bit number using Shors algorithm on a quantum computer would take one day. Factoring the same number using a classical computer would take a data center the size of Germany one hundred years to complete [5].

Quantum computing promises breakthroughs in fields that deal with large data sets and require massive amounts of processing power such as genetics, molecular modeling, astrophysics, chemistry, data science, artificial intelligence, and cryptography. This expanding field will need many trained professionals proficient in both physics and computing. This study will document the student experience using the IBM Q Experience and QISKit to pursue training in programming for quantum computers.

2 Study Overview

This study explores the educational material available online for students interested in programming for quantum computers focusing on the interface offered by IBM that allows users to run code on publicly available quantum computers or quantum simulators.

The study uses IBM Q Experience to create two simple quantum programs and documents the experience working with the development kits, programming languages, user interfaces, online documentation, and tutorial information. The study utilizes IBMs QISKit, a Python based development kit, to create a simple random password generation program and a card game program that runs on IBMs publicly available quantum computer in Yorktown, New York.

The goal of this study is to produce an informational review of a students early experience with quantum computing focusing on improving the currently available material in order to attract more students to the growing field. Toward this goal, the study includes a supplemental quantum tutorial using Jupyter Notebook that incorporates the example programs and serves as a guide for students interested in quantum computing.

3 Literature Review

In 1965, Gordon Moore, the co-founder of Fairchild Semiconductor and Intel proposed that transistors (the main component of central processors) would keep shrinking every year due to engineering advancements in the semiconductor industry. Specifically, Moore claimed that the industry would double the number of transistors in CPUs approximately every year. He would later revise this to every 2 years in 1975, while David House, an Intel executive at the time, noted that the computing power would also double every 18 months [6]. In the late 1950s, some chips contained 200 transistors; and by 2005, "Intel would produce chips with 1 billion transistors [6]." The semiconductor revolution gave rise to the internet, smart phones, and the Internet of Things.

Consumer demand for more powerful devices has increased, and manufacturers strive to push the limits of Moore's law. However, simple physics dictates that transistors can only be so small. "No physical quantity can continue to change exponentially forever," Moore said in 2003 [6]. In March of 2016, the semiconductor industry would formally acknowledge the nearing end of Moore's law [7]. This was due in part to the excess heat generated by densely packed silicon circuity in small devices, and that when transistors shrink to only 10 atoms across, "electron behavior will be governed by quantum uncertainties that will make transistors hopelessly unreliable [7]." Researchers hope that quantum computing will allow a new phase of exponential growth in computing speeds. IBM announced in a press release in May of 2016, "with Moore's Law running out of steam, quantum computing will be among the technologies that could usher in a new era of innovation across industries [8]."

In his influential paper, "The Physical Implementation of Quantum Computation" David DiVincenzo proposed five criteria that quantum computer implementation must meet. First, the system must utilize qubits for making computations. Second, the system must have the ability to initialize the qubits to a known state. Third, the computer must use a universal set of quantum gates. Fourth, the gates must operate faster than the information stored in the qubits can be lost due to interaction with the surrounding environment. And finally, measurement of information contained in qubits must be possible [9]. A machine that meets these criteria is classified as a quantum computer.

The first quantum computer was built at Oxford University in 2004. The machine was a nuclear magnetic resonance (NMR) machine that was able to solve simple calculations twice as fast as a classical computer. NMR machines utilize magnetic field emissions created by electrons orbiting a nucleus and radio waves of varying frequencies to manipulate the electron fields causing the electrons to act as qubits [3].

In the last five years, large corporations such as Google and IBM, and start ups such as Rigetti and Quantum circuits have had major success implementing superconductor quantum computers. Superconductor computers require temperatures of millikelvin to operate. Qubits can store information through the charge of the particles. Superconductor machines use pulses of radio frequencies to control qubits in order to execute quantum operations [3].

Researchers are also working on other implementations of quantum computers. Linear optic machines have potential for future breakthroughs due to low infrastructure cost. Diamond machines use impurities in diamonds to capture single electrons and use them as qubits. Diamond machines have the advantage over superconductors in that they can operate at 4 K. The potential benefits of Diamond implementations are promising, however the diamond is not as well researched as other technologies.

The cutting edge of quantum computer implementations is the anyon computer. Researchers theorize that "under some circumstances certain material can behave as if they are holding particles that do not actually exist, known as quasi-particles [10]." These quasi-particles, or anyons, can be used as qubits. The physical properties of anyons protect from decoherence and give anyon machines an advantage over other implementations. Anyon research is promising although still in the theoretical phase.

There are many engineering challenges in quantum computing. As mentioned above, decoherence describes a process in which information stored in qubits is lost due to its interaction with the environment. All implementations of quantum computers discussed above suffer from decoherence. Decoherence must be accounted for using quantum error correction (QEC). QEC is necessary for accurate calculations but drastically increases the necessary number of qubits required for computations [5].

Quantum computers must also be capable of fault tolerant (FT) computations. Fault tolerance means that small errors do not lead to information loss or larger errors in calculations. C. G. Almudever et al. suggest that a number of qubits in the millions may be required to perform QEC and FT computations successfully [5].

Another challenge is that quantum computers are expensive, large, and difficult to maintain. For most organizations, it is impractical to implement a machine with even a small number of qubits. Cooling the quantum machine accounts for much of the cost and difficulty of operation. Similar to classical computers, quantum computers require cooling, but at much lower temperatures [11]. When a qubit changes its state it generates heat, and because qubits change states often during program execution they can generate massive amounts of heat. Cooling quantum superconducting processors requires extremely low temperatures. For example, at D-Wave Systems in British Columbia Canada, the cryogenic system processor operates at 15 mK, "approximately 180 times colder than interstellar space." In other words, the quantum processor requires $-460\,^{\circ}$F or $-273\,^{\circ}$C to operate [12].

Due to massive operational costs and engineering challenges, R. Van Meter and S. J. Devit have proposed two new criteria to be added to DiVincenzo's original quantum criteria. First, that quantum systems be small, cheap, and reliable enough to be practical and fast enough to make using one worthwhile. Second, that "implementation limitations make locally distributed computation imperative, which requires system area networks that are fast, high-fidelity, and scalable [10]". These two criteria have led to research into distributed quantum systems. If one organization is able to maintain a 20 qubit machine, another organization a 15 qubit machine, and a third organization a 12 qubit machine, the ability to use all three machines simultaneously would allow the quantum computation to access 47 qubits. Researchers are currently working on connecting and scaling these machines.

The complexities of creating distributed quantum systems show the necessity for attracting students and professionals from multiple disciplines to the field. Researchers skilled in multiple disciplines of physics are needed to develop quantum chips. Electrical engineers and computer architects must develop quantum instruction set architectures and error correction schemes. Computer scientists must develop quantum compilers and higher level programming languages capable of running on quantum hardware. Chemists and materials engineers must develop hardware capable of maintaining the quantum states of qubits and extremely low temperatures. Quantum computing sits at the crossroads of these disciplines. Attracting skilled and knowledgeable individuals to the field of quantum computing is vital to its future success.

4 Methodology

This study utilizes IBMs QISKit, a Python based software development kit to implement two quantum computing examples. The QISKit requires Python 3.5 or later and runs on Windows, OSX, and Linux. Users who register with IBM can gain access to the Q Experience API which allows users to run code on a publicly accessible quantum computer in Yorktown, NY. Currently, IBM allows access to three quantum chips. IBMQX2 and IBMQX4 are both 5 qubit chips. IBMQX5 is a 16 qubit chip [13]. The QISKit also allows users to run quantum simulations on a local machine.

The basic structure of a quantum program is organized into three steps: build, compile, and run. In the build step, the user creates a quantum circuit composed of quantum registers. The user can then add quantum gates to manipulate the registers. Gates are essential to quantum programs. They perform operations directly on qubits. In the compile step, the user selects a back-end, either their local machine or a publicly available IBM quantum chip, on which to execute their quantum code. In the final step, the user runs the code and receives a result. The user can select various options that can affect execution in the step.

All quantum programs operate on qubits using gates. A single qubit is represented visually using a Bloch Sphere.

Figure 1 on page 553 is an illustration of a Bloch Sphere. The Bloch Sphere has a radius of 1 and a vector from its center to its surface. The state of the qubit is represented by the point where the vector meets the surface of the sphere. The point at the top of the sphere represents $|0\rangle$ and the point at the bottom of the sphere represents $|1\rangle$. Gates manipulate the position of the qubit on the surface of the sphere to perform calculations [4].

This Bloch Sphere is a representation of a qubit with X, Y, and Z axis labeled. $|0\rangle$ at the top of the sphere. $|1\rangle$ at the bottom of the sphere. The position of the qubit on the surface of the sphere is represented by the solid orange vector. Angle θ represents a state of superposition when the value of the qubit is between $|0\rangle$ and $|1\rangle$. Angle ϕ represents rotation around the Z axis or phase of the qubit.

The quantum gate is the primary tool a developer can use to interact with qubits. Therefore, in order to write a quantum program, a developer must have a thorough understanding of how to use quantum gates. The following sections will explain in detail the two quantum computing program examples produced by this study including how they make use of quantum gates [14].

5 Program I - Quantum War

The first programming example is the quantum version of the children's card game "War". This program demonstrates a practical application for the quantum principle of superposition. Figure 2 on page 554 shows how a small number of qubits can do the work of many classical bits by using a real-world example to increase comprehension. It also illustrates how to create and measure two quantum circuits.

To play the game, a deck of playing cards is divided evenly among two players. Each player simultaneously reveals a card, the player with the card of the highest order wins the battle. The winning player claims the losing player's card. The first player to run out of cards loses. After drawing, the cards are removed from the deck. Traditionally, the rules of War state that the losing player's cards are added to the winning player's deck. This study has opted to forego this portion of the rules to keep the example simple for those first learning about quantum

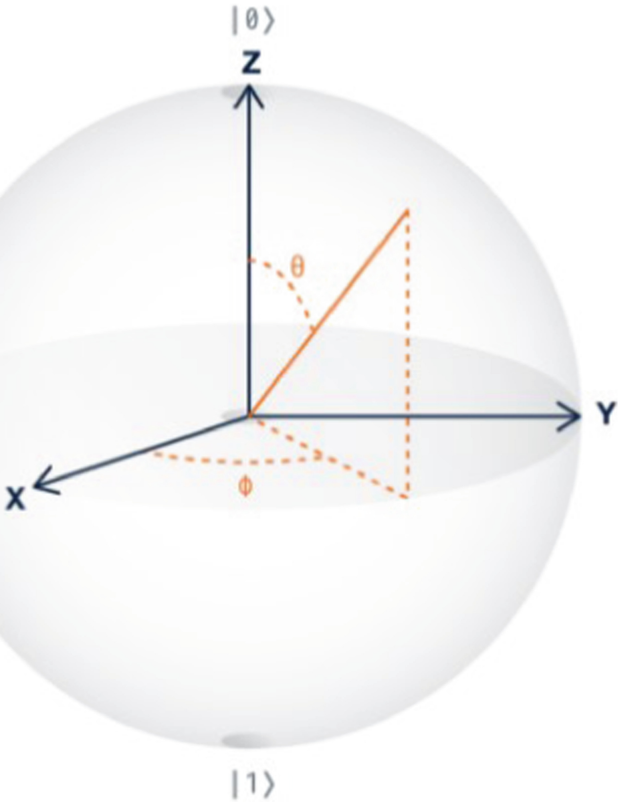


Fig. 1. Bloch sphere [4]

programming. This study has also opted to omit the three card W-A-R draw at the start of each round for simplicity.

A traditional deck of cards contains fifty-two unique cards. In order for a classical computer to represent one card of the fifty-two options, six bits are required as illustrated in Fig. 2. In comparison, a classical computer requires three hundred and twelve bits (six bits per card) to represent all fifty-two options simultaneously. The principle which allows the quantum computer to represent all fifty-two options with only these six qubits is superposition. Superposition, illustrated in Fig. 3 on page 554, is when the value of a qubit is a linear combination of both $|0\rangle$ and $|1\rangle$. This means that the qubit represents all possible values simultaneously. Thus, six qubits, all in a state of superposition, represent all fifty-two possibilities when drawing from a deck of playing cards.

Another means of demonstrating the position of the qubits as well as the gates influencing them is by creating a simple quantum circuit schematic as illustrated in Fig. 4 on page 555. For this program, each deck is configured in the same fashion and can be represented with a single schematic. The schematic allows

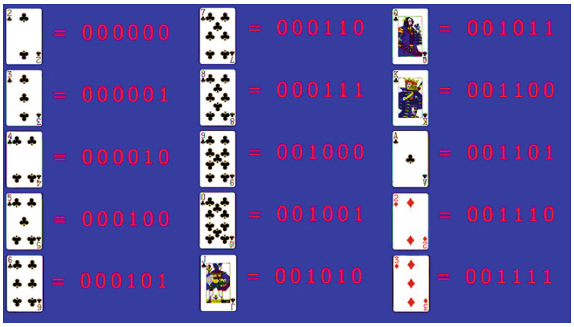


Fig. 2. Visual representation of how cards are identified in the quantum deck

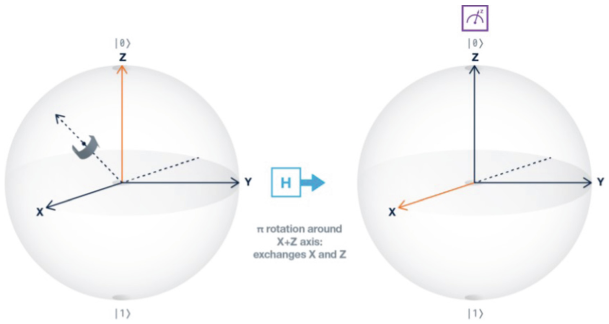


Fig. 3. Bloch sphere representing H gate rotation around the X + Z axis. The qubit, represented by the orange arrow, begins with a state of $|0\rangle$. After the H gate performs the rotation, the qubit is in a state of superposition. The value of the qubit is a linear combination of $|0\rangle$ and $|1\rangle$ [4]

the observer to see the qubits, how they are manipulated, and the projected outcome based on the gates applied to each qubit.

This study slightly modifies the rules of War to illustrate more fully the quantum principles at play. Each player plays with a full deck of fifty-two cards. Each deck is represented by six qubits, twelve qubits in total.

The program begins with two classes that are used to keep track of the cards in each player's deck. The Card class holds information about each playing card including, the suit, name, and value of the card. The Deck class contains logic for creating a deck of cards containing 52 cards with values 2–10, Jack, Queen, King and Ace with suits Clubs, Diamonds, Hearts, and Spades. It also contains the logic for drawing a card and removing it from the deck.

Following the class definitions, the user chooses to run the program on a local quantum simulator or IBM's IBMQX5, a 16 qubit chip. The simulator runs faster than making a connection to the IBMQX5 and is guaranteed to produce a result. Due to high volume of usage, a request to use the IBMQX5 can sometimes timeout.

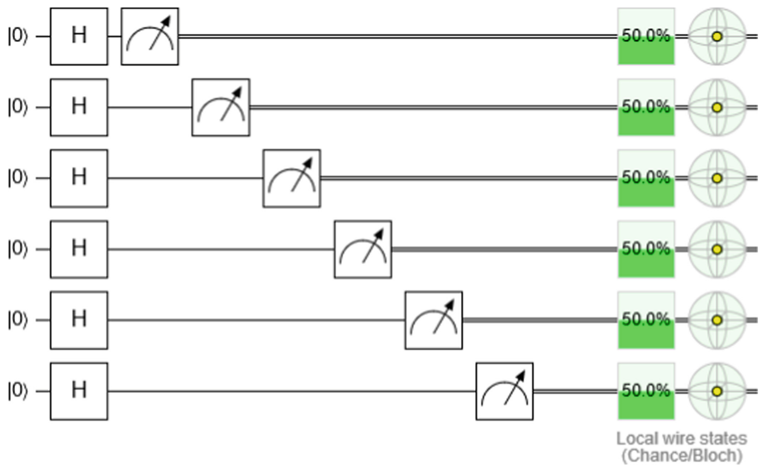


Fig. 4. Visual representation of quantum circuit schematic. Each horizontal line represents one qubit. Along the line gates manipulate the value of the qubit. After the gate manipulate the value of the qubit, the program measures the value of the qubit, represented by the arrow shaped measurement indicator. On the right side, the percentage represents the probability that the qubits value is $|1\rangle$. On the far right, a Bloch sphere represents the qubit with the gate(s) applied

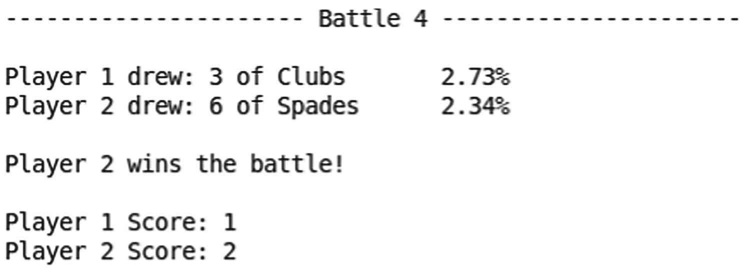
Next, the program creates two quantum circuits to represent each deck. Each circuit contains six qubits. All six qubits have an H gate applied to them to induce a state of superposition represented in Fig. 3 on page 554. This will allow all six qubits to potentially represent any of the 52 cards in the deck. The program now contains two quantum circuits, each of which represents a deck of 52 playing cards.

Next, the battle between players begins. Player One's quantum circuit is measured. This represents drawing a card from the top of the deck. Then, player Two's quantum circuit is measured. The measurement returns the probability of drawing any one of the fifty-two cards from the top of the deck. The card drawn is determined by the measurement taken of the state of the qubits.

The six qubits representing each deck are measured 1024 times. One measurement is referred to as a "shot."In a single shot, the probability that a qubit is equal to $|0\rangle$ or $|1\rangle$ is measured and recorded. After all 1024 shots, a dictionary is returned that contains the number of times all sixty-four combinations of the six qubits were measured. The value between 000000 and 110011 that is measured with the highest frequency corresponds to the card that the player drew. The drawn card is then removed from the deck and cannot be drawn again. Finally, Python logic compares the two cards and determines a winner for the battle.

The program then presents the players with the cards that they drew and the likelihood that the qubits represented that card at the time of measurement. The likelihood is calculated by dividing the number of times the value was measured by the total number of shots and represented as a percentage in the output. Sample output is presented in Fig. 5 on page 556.

The program is successfully able to simulate a truly random event. In classical computing, random functions are usually pseudo random. The functions must be seeded with the date and time or other information to produce a result that seems random but is not actually random. In contrast, the behavior of the qubits is truly unpredictable. Thus, when measuring qubits to represent a real world activity, such as drawing a card from a shuffled deck, the result is truly random. The card game simulation employs qubits in a practical situation that requires truly random results.



```
--------------------- Battle 4 ---------------------

Player 1 drew: 3 of Clubs       2.73%
Player 2 drew: 6 of Spades      2.34%

Player 2 wins the battle!

Player 1 Score: 1
Player 2 Score: 2
```

Fig. 5. Sample output of the quantum war program. Player 1 drew the 3 of clubs. This draw was measured in 2.73% of shots. Player 2 drew the 6 of Spades which was measured in 2.34% of the shots.

6 Program II - Random Password Generator

The second programming example is the quantum version of a random password generator. This program demonstrates how to use superposition to generate ASCII characters. It also shows how a small number of qubits can do the work of many classical bits and uses a practical example to increase user engagement. It also shows how to use a single measurement to get multiple results and illustrates how to account for noise when measuring a quantum circuit.

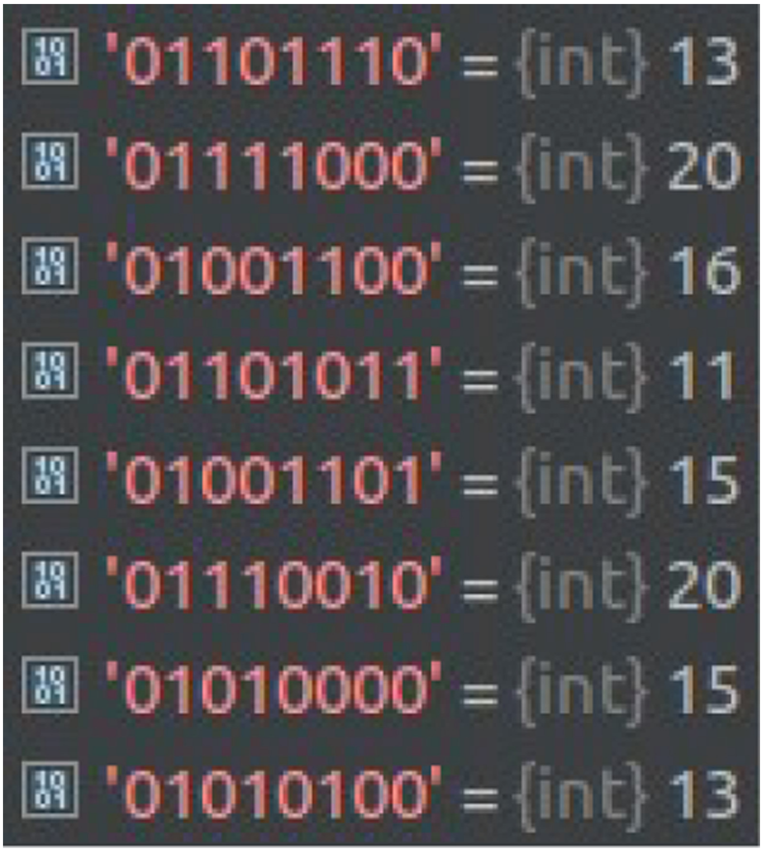
This program uses a quantum circuit to create a truly random password using English ASCII characters. The program uses an 8 qubit quantum circuit to generate a variable length random password. A single ASCII character, represented on a classical computer, uses 8 bits. In the classical implementation, 8 bits represents a single character, and only that character. In order for the classical machine to represent a different character, the value of one or more bits must be changed or more bits must be used. The quantum implementation can utilize superposition to create multiple characters using only 8 qubits by measuring each qubit multiple times. Thus, the quantum circuit represents all ASCII characters simultaneously when in a state of superposition.

The quantum computer has two advantages over the classical computer when it comes to password generation. The first advantage is that because quantum computers use qubits with superposition, they can generate longer passwords in less time than a classical computer. The second advantage is that the quantum computer is capable of generating a truly random password while the classical computer is not. While the classical computer utilizes pseudo random functions to produce results that seem random, only qubits can produce truly random results. As with the Quantum War program, the motion of quantum particles is unpredictable. Thus, using qubits can produce a password that is truly random and thus stronger than a random password generated on a classical computer.

The program begins by creating an 8 qubit quantum circuit. Next, the program uses gates to set the qubits to the proper states for execution. All English ASCII characters begin with the leading bits 01. The following six bits are variable and can be set to either 0 or 1 depending on the character being represented. Thus, to represent an ASCII character with qubits, the leading two qubits, qubits 7 and 6, must be set to 0 and 1 respectively. These values must not change during the execution of the program or non-English characters will appear in the results. The program achieves this by leaving qubit 7 in its initialized state of 0 and using an X gate to set qubit 6 to 1. Next, qubits 0 through 5 are put into a state of superposition using H gates.

This is demonstrated in the quantum circuit schematic in Fig. 4 on page 555. This program is a simple yet efficient representation of the randomness of superposition and demonstrates the advantages over classical computing. When all eight qubits are set to the correct values, the program measures the quantum circuit. This determines the characters that are used to generate the password. By measuring the qubits, the program can determine the probability that they represent a particular character at the time of measurement. The eight qubits representing a single character are measured in 1024 shots. Each shot is a single measurement that represents a single character. After all 1024 shots occur, QISKit returns a dictionary that contains the frequency that each character was measured. Figure 6 on page 558 illustrates a dictionary representation of the measurement results.

Because Quantum measurements can be noisy, the program must also account for noise in the measurement. Noise is unintended environmental interference that may distort the results of a measurement. To account for this, the program eliminates all results that were measured in less than 2% of shots. The 2% percent noise threshold is recommended in several QISKit examples. The program assumes that any character which appears in greater than 2% of shots is a legitimate qubit value and not noise. In Fig. 6 on page 558, the results of 1024 shots are displayed. The character 01101110, the character n, was measured in 13 shots. To determine if the character n was actually measured or is the result of noise, the program divides 13 by 1024 which equals 0.0127. Since the character n only appeared in 1.2% of shots, it is indistinguishable from noise and thus is not included in the password. This noise threshold check is performed for each

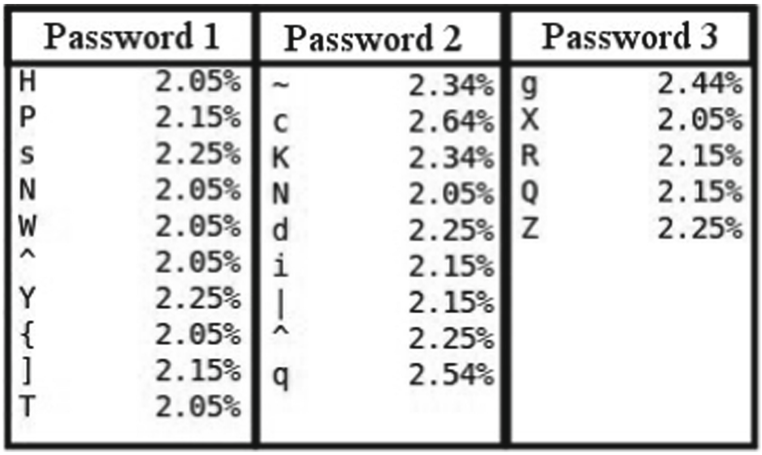


```
'01101110' = {int} 13
'01111000' = {int} 20
'01001100' = {int} 16
'01101011' = {int} 11
'01001101' = {int} 15
'01110010' = {int} 20
'01010000' = {int} 15
'01010100' = {int} 13
```

Fig. 6. An excerpt from the dictionary representing the frequency that each character was measured during each of the 1024 shots. On the left, is the binary representation of the character and on the right the number of shots which the 8 qubits represented that character. The first character in the dictionary 01101110 is the character n and it was measured in 13 of the 1024 shots

measured character. All characters that the program measures in more than 2% of shots are added to the password.

Because the program accounts for noise, the final length of the password is variable. Generally the password is between six to eight characters but may be longer or shorter. This is due to the random nature of qubits and the fact that noise can distort measurements. Sometimes the measurement frequency of a few characters rises above the noise threshold. Other times, ten or more characters may be measured over 2% of the time. Figure 7 on page 559 demonstrates three sample passwords generated by the quantum program. Each column represents a single password. On the left is the character, and on the right the percentage of shots that it was measured in. Running the program multiple times will predictably produce varied results.



Password 1		Password 2		Password 3	
H	2.05%	~	2.34%	g	2.44%
P	2.15%	c	2.64%	X	2.05%
s	2.25%	K	2.34%	R	2.15%
N	2.05%	N	2.05%	Q	2.15%
W	2.05%	d	2.25%	Z	2.25%
^	2.05%	i	2.15%		
Y	2.25%	|	2.15%		
{	2.05%	^	2.25%		
]	2.15%	q	2.54%		
T	2.05%				

Fig. 7. Three output examples of the quantum random password generator. Each character of the password is displayed vertically with the probability that the character was measured displayed across from it. The passwords are variable length depending on how many characters were measured more than 2% of the time to account for noise.

This program can create randomized passwords that meet acceptable password guidelines. The passwords are truly random, use upper and lowercase English characters and most commonly used special characters. Similar to the last program, it illustrates the concepts of superposition and uses qubits to represent multiple outcomes simultaneously. In addition to having real world application, the combination program can also help students understand intermediate quantum concepts like noise, H gates and X gates.

7 Results

This study has successfully developed two working quantum programming examples that return valid output. Both programs illustrate the quantum concepts of superposition and how to utilize gates to perform operations on qubits. The programs also demonstrate the efficiency of performing calculations using qubits rather than using classical bits.

The first program, Quantum War, shows how qubits can simulate a tangible activity such as drawing from a deck of cards. The program runs on the local simulator as well as the 16 qubit IBMQX5 machine available over the internet. The program is more economical in its representation of cards than a classical machine and each card drawn is truly randomized. Figure 5 on page 556 represents a single output example showing a single battle in which each player has drawn a card from a randomized quantum deck.

The second program, Quantum Random Password Generator, successfully produces truly random passwords. It uses all English ASCII characters, both uppercase and lowercase, and commonly used special characters. In 98% of attempts the program generates a password of 7–14 characters in length, which will satisfy most password requirements. The program can be easily adjusted to increase or decrease the quantity of characters within the output. The program runs on the local simulator as well as the IBMQX5 quantum machine. Figure 7 on page 559 represents three output examples from the Quantum Random Password Generator as it is currently configured.

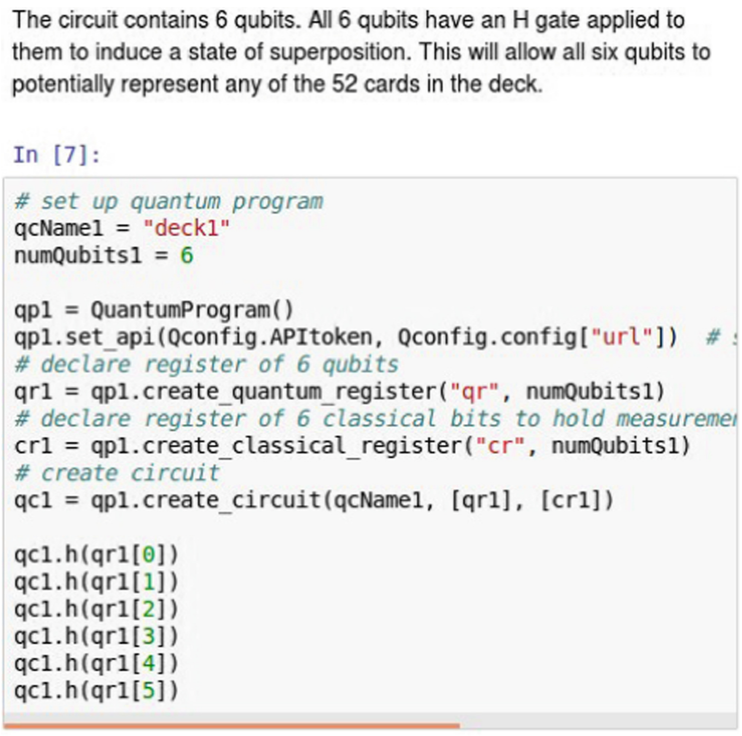
This study utilized various IBM Q Experience tutorials and the QISKit software development kit to create a development environment using Python 3.5, Conda, and Jupyter Notebook. The IBM tutorials were student friendly, however the difficulty increased significantly after introduction to the basics. The IBM QISKit tutorials require more intermediate level examples that illustrate simple operations using quantum gates with less abstract functionality. This study offers two such practical examples, Quantum War and Quantum Random Password Generator, to the growing body of literature. This study also produced Jupyter Notebook pages that accompany each program and document how the program operates. Jupyter Notebook is an integrated development environment (IDE) that allows developers to create a document that contains runnable code and formatted narrative text tutorials. The notebook pages that accompany each program were written to match the style and substance of existing IBM QISKit tutorials.

The notebooks will serve two purposes. First, they organize the code into understandable blocks and explain how each block functions. This approach is preferable to in-line comments because they are clearer, more visually appealing, easier to understand, and can contain explanations of concepts not normally found in in-line comments. Figure 8 on page 561 provides an example notebook page.

Overall, this study found that Jupyter Notebook pages are better for the student learning experience than parsing in-line comments or reading documentation and code in separate documents. The second purpose of the notebooks is to serve as a tutorial for students interested in quantum computing who have read and understood basic quantum programming examples, but are not ready for more advanced material. This study found that more intermediate material of this nature was needed. As mentioned in the last paragraph, there is a large gap between introductory material and advanced material. This study provides two ready now working intermediate programming examples that can help students bridge the gap from basic quantum programming to understanding advanced examples.

This study has contributed two intermediate quantum programming examples for students interested in advancing their quantum programming knowledge.

Overall, the student quantum experience was a positive one. IBMs tutorials and visual aids to familiarize users with both the programming environment and underlying physical principles are well-crafted and understandable. IBM has clearly documented the functionality of the QISKit on the Q Experience website. IBMs choice to write the QISKit in Python is advantageous because Pythons easily readable syntax clearly demonstrates quantum concepts such as



The circuit contains 6 qubits. All 6 qubits have an H gate applied to them to induce a state of superposition. This will allow all six qubits to potentially represent any of the 52 cards in the deck.

In [7]:

```
# set up quantum program
qcName1 = "deck1"
numQubits1 = 6

qp1 = QuantumProgram()
qp1.set_api(Qconfig.APItoken, Qconfig.config["url"])  # :
# declare register of 6 qubits
qr1 = qp1.create_quantum_register("qr", numQubits1)
# declare register of 6 classical bits to hold measureme
cr1 = qp1.create_classical_register("cr", numQubits1)
# create circuit
qc1 = qp1.create_circuit(qcName1, [qr1], [cr1])

qc1.h(qr1[0])
qc1.h(qr1[1])
qc1.h(qr1[2])
qc1.h(qr1[3])
qc1.h(qr1[4])
qc1.h(qr1[5])
```

Fig. 8. An excerpt from a Jupyter notebook page. This excerpt is from the quantum war program and shows how to create and initialize a quantum circuit. At the top of the figure is the narrative explanation and below a neatly separated block of code that can be run from within Jupyter notebook.

initializing a quantum circuit, applying gates to qubits, and printing the results in a classical format. The only negative of note is that it can be difficult to find clear explanations that bridge the gap between intermediate and advanced quantum topics such as QRAC (Quantum Random Access Code) and Phase Gates.

8 Conclusions

The two examples presented in this paper are available to the researchers at IBM for use in the QISKit tutorial materials. These concrete and familiar examples - a card game and random password generator - have the potential to enrich a students introduction to quantum programming and can assist in filling the gaps between beginner and advanced learning materials.

The future for quantum computing is bright. However, there are still hurdles that must be cleared. High cost, poor error correction, limited access to existing quantum machines for learning and experimentation, and untrained personnel still stymie development and adoption. While there is a wealth of introductory material available on the web from various publications, news outlets, and educational institutions that covers the background information of quantum computing and quantum mechanics, little content exists outside of IBM QISKit that clearly explains programming for quantum computers. For instance, currently there are none or few tutorials for quantum programming found on popular programming learning sites like Udemy, Udacity, Coursera, and Youtube. As companies such as IBM, Google, and Microsoft scale up their quantum computing research, there will certainly be a need for more quantum programmers.

As argued by R. Van Meter and S. J. Devitt, affordability and scalability of quantum implementations will increase their usefulness in a broad range of fields, including biophysics and security [10]. As the usefulness of quantum computers increases, their availability will increase. As their availability increases, the need for trained quantum programmers will also increase. This study offers simple, high-yield student-to-student teaching examples for introductory courses in quantum computing. Helping students develop literacy in programming for quantum computers will depend on educators who can create clear, applicable, and engaging teaching materials that will attract motivated programmers to the field and help usher in the next revolution in computing.

References

1. Lyshevski, S.E., Iafrate, G.J., Brenner, D., Goddard III, W.A.: There's plenty of room at the bottom: an invitation to enter a new field of physics. In: Handbook of Nanoscience, Engineering, and Technology, 2nd edn., pp. 27–36. CRC Press (2007)
2. Arons, A.B., Peppard, M.B.: Einstein's proposal of the photon concepta translation of the annalen der physik paper of 1905. Am. J. Phys. **33**(5), 367–374 (1965)
3. Kumar, K., Sharma, N.A., Prasad, R., Deo, A., Khorshed, M.T., Prasad, M., Dutt, A., Ali, A.B.M.S.: A survey on quantum computing with main focus on the methods of implementation and commercialization gaps. In: 2015 2nd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE), pp. 1–7. IEEE (2015)
4. IBM Q Experience. https://quantumexperience.ng.bluemix.net (2018)
5. Almudever, C.G., Lao, L., Fu, X., Khammassi, N., Ashraf, I., Iorga, D., Varsamopoulos, S., Eichler, C., Wallraff, A., Geck, L., et al.: The engineering challenges in quantum computing. In: 2017 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 836–845. IEEE (2017)
6. Kanellos, M.: Moores Law to Roll on for Another Decade. CNET News.com (2003)
7. Waldrop, M.M.: More than moore. Nature **530**(7589), 144–148 (2016)
8. Vu, C.: IBM makes quantum computing available on IBM cloud to accelerate innovation. IBM News Room (2016)
9. DiVincenzo, D.P.: The physical implementation of quantum computation. Fortschritte der Physik: Prog. Phys. **48**(9–11), 771–783 (2000)
10. Van Meter, R., Devitt, S.J.: The path to scalable distributed quantum computing. Computer **49**(9), 31–42 (2016)
11. Tan, K.Y., Partanen, M., Lake, R.E., Govenius, J., Masuda, S., Möttönen, M.: Quantum-circuit refrigerator. Nat. Communi. **8**, 15189 (2017)

12. Lee, P.: Dwave Quantum Computing Company. News Room (2018)
13. (Ginni) Rometty, V.M.: IBM QISKit Github Project. IBM. https://github.com/QISKit (2018)
14. Barenco, A., Deutsch, D., Ekert, A., Jozsa, R.: Conditional quantum dynamics and logic gates. Phys. Rev. Lett. **74**(20), 4083 (1995)