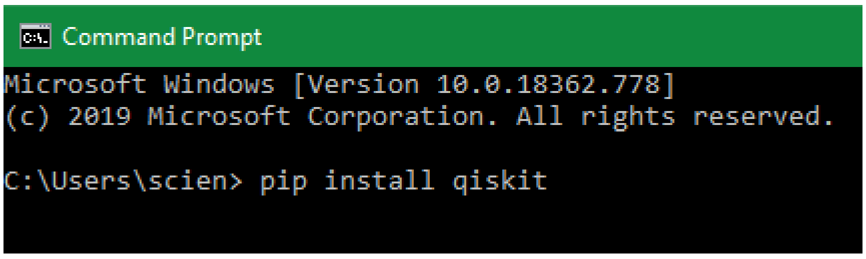
Introduction to Quantum Programming with Qiskit

Abdulah Amer

1 First Things First

We first begin with setting up our environment to work with Qiskit. You will need to install Python 3 from here. We will then use command line to install Qiskit. This can be done using the 'pip install qiskit' command in the command line[1] just as seen below.



```
Command Prompt
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\scien> pip install qiskit
```

Figure 1: pip makes installing stuff easy using the commandline

Now, we need to install our environment, Jupyternotebooks. This can be done easily now that we have python by using another command in the commandline, 'pip install notebook'. After that is finsihed downloading we now have the 'jupyter notebook' command. This will start a python kernel[2] and open the Jupyternotebooks homepage. On the right hand side of the screen there is a 'new' drop down menu, click on the Python 3 notebook option and you should see a screen like the one on top of the next page.

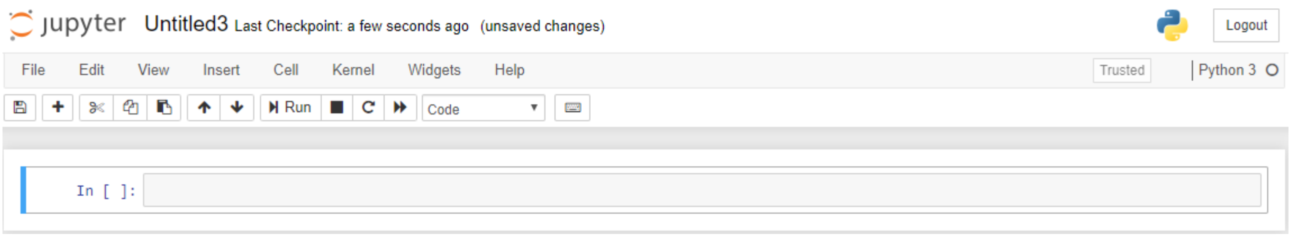


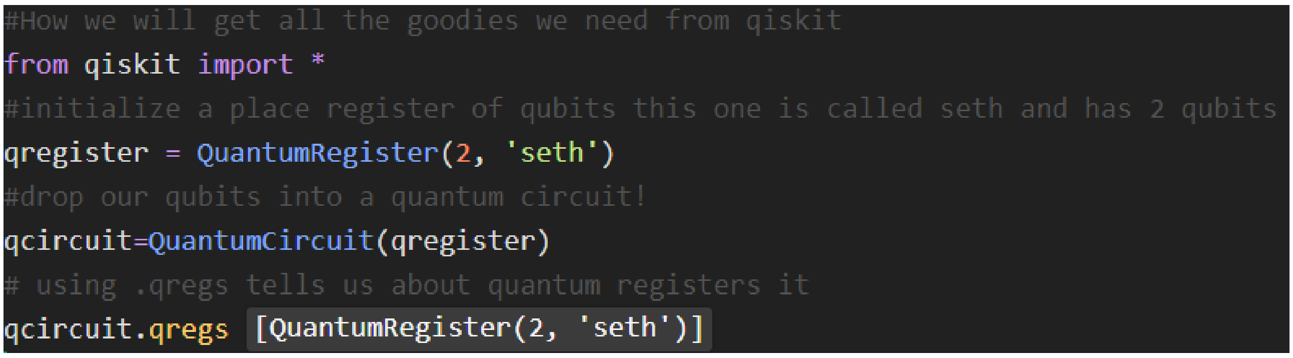
Figure 2: Jupyternotebook Cell

Lets get started by coding up our first QuantumCircuit! The QuantumCircuit Object in Qiskit is the bread and butter of the package. We start by initizalizing a quantum register, a place that says this is how many qubits I have, and then putting our quantum register into our QuantumCircuit.

In this order, we will import qiskit into Python so we can use it. Define a quantum register with 2 qubits and give it a name. Define a quantum circuit that will use our 2 qubits. Then get some information about the quantum register used in our new quantum circuit. To run

[1]for mac users commandline just means terminal
[2]tastier than corn kernels!

our code you either just click on a cell you want to run and click the 'Run' button, or click on 'cells' and click 'run cell' or 'run all'.

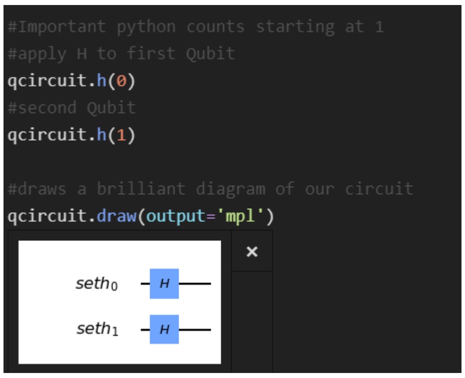


```
#How we will get all the goodies we need from qiskit
from qiskit import *
#initialize a place register of qubits this one is called seth and has 2 qubits
qregister = QuantumRegister(2, 'seth')
#drop our qubits into a quantum circuit!
qcircuit=QuantumCircuit(qregister)
# using .qregs tells us about quantum registers it
qcircuit.qregs [QuantumRegister(2, 'seth')]
```

Figure 3: Importing Qiskit

2 Everyone's Favorite Quantum Gate

Now we will get to the buisness of applying gates to our quantum circuit. The First gate we will use is the Hadammard Gate. This is implemented by the following



```
#Important python counts starting at 1
#apply H to first Qubit
qcircuit.h(0)
#second Qubit
qcircuit.h(1)

#draws a brilliant diagram of our circuit
qcircuit.draw(output='mpl')
```

Figure 4: The humble Hadammard gate

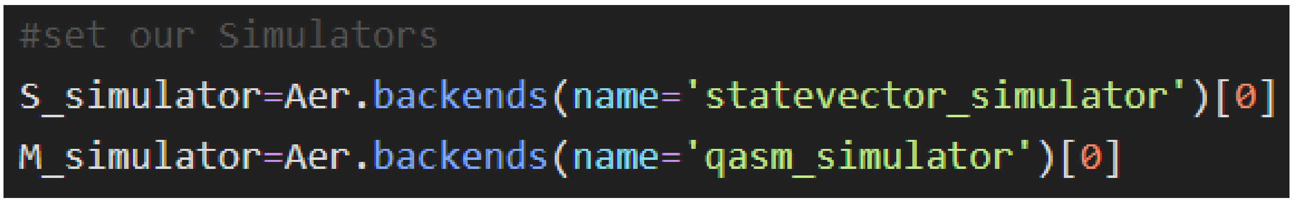
We could also just use qcircuit.draw() but that picture would not look nearly as neat. Now we have our gates applied lets take a look at making measurements using qiskit. We will need a little more setting up however. Making sure we define our simulator backends.[3]

Qiskit works by providing you with a simulator that is able to run the experiments that we want to run. We need in particular a Statevector simulator and a Measurement simulator [4]. We will call them S_simulator and M_simulator respectively. We do this by simply making variables for the pair.

Now we have tools to make measurements and to check the states of our quantum system! The first thing we would like to do is get a statevector that reptresents our quantum circuit. This is done by executing a job on the appropriate simulator. Lets make a vairable called 'job' and execute a job on the statevector simulator, then get the result of that job and print it out for us to see.

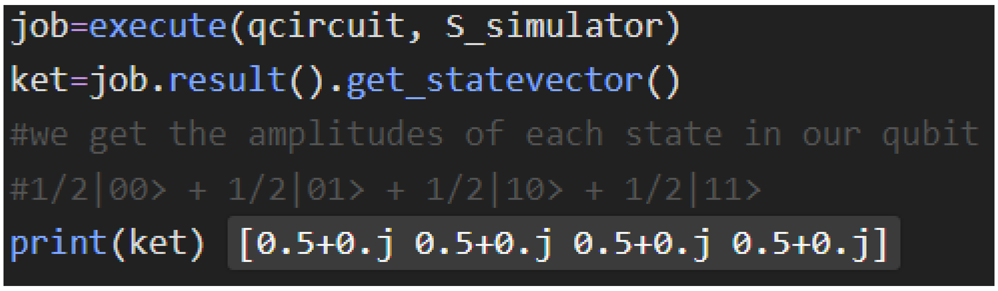
[3]Do not worry about what a backend is we are just gonna write them in

[4]Actually a QASM simulator that reads our instructions and uses those to do measurements in the simulator



```
#set our Simulators
S_simulator=Aer.backends(name='statevector_simulator')[0]
M_simulator=Aer.backends(name='qasm_simulator')[0]
```

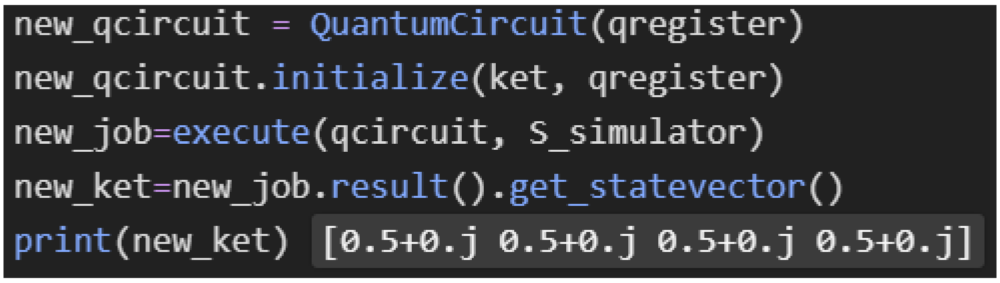
Figure 5: setting up simulators



```
job=execute(qcircuit, S_simulator)
ket=job.result().get_statevector()
#we get the amplitudes of each state in our qubit
#1/2|00> + 1/2|01> + 1/2|10> + 1/2|11>
print(ket) [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
```

Figure 6: Our first ket

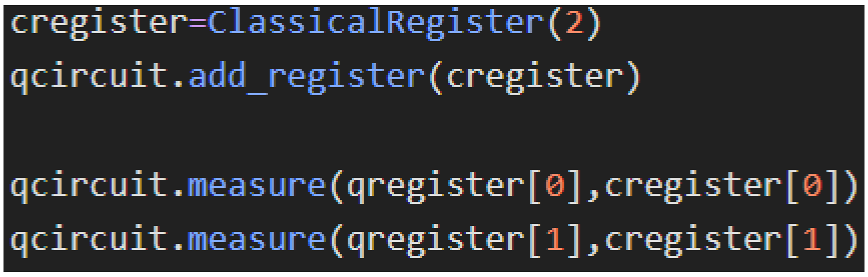
We can even take this state we have a put into a new quantum circuit. This is done by using the '.initialize' method. This can be useful for setting up a bell state at the beginning of an experiment for entanglement or to reset a state to make repeated measurements on the circuit.



```
new_qcircuit = QuantumCircuit(qregister)
new_qcircuit.initialize(ket, qregister)
new_job=execute(qcircuit, S_simulator)
new_ket=new_job.result().get_statevector()
print(new_ket) [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
```

Figure 7: initialization

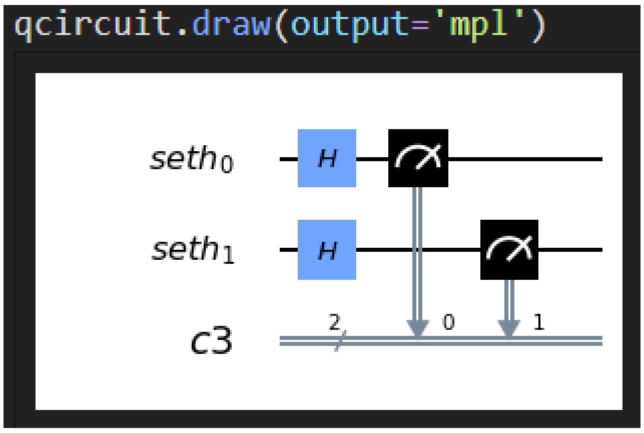
We now introduce the classical register. Much like its name suggests, it is just a place that says I have some number of classical bits. Classical bits are used with their quantum cousins to hold results from quantum experiments. To measure both qubits we will need 2 classical bits. We begin by creating a classical register and then adding it to our quantum circuit. We make measurements by specifying the qubit we are measuring and the classical bit we are putting the result into.



```
cregister=ClassicalRegister(2)
qcircuit.add_register(cregister)

qcircuit.measure(qregister[0],cregister[0])
qcircuit.measure(qregister[1],cregister[1])
```

Figure 8: Making measurements

We can draw our circuit now to see how it looks just because it really is neat.

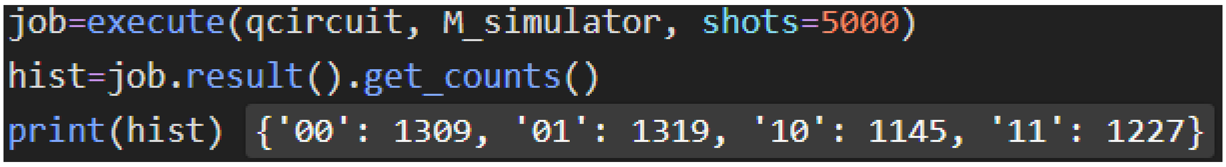


```
qcircuit.draw(output='mpl')
```

Figure 9: neat

Since we are in buisness of measuring we need to use the measurement simulator. Very similar to the way we executed a job to get the statevector we will do the same to get the measurement. Qiskit allows us however to do multiple runs of a measuremnt on a state, each run is called a shot. Each execution of a measuremnt job will by default to 1024 shots but we can also tell Qiskit how many shots we want to do for a particular measurement. We can even get a really neat histogram to display the results.

What else can we do with qiskit? There are other visualizations we can do. Including making bloch vectors on a bloch sphere and showing how the bloch vector rotates as we move it around the sphere. We can run more complicated circuits using all our friends, the Pauli matrices and CNOT among others. You can see that you can take any interesting circuit you have seen in class and type it up and see how it behaves and measure it and see statevectors! For more information do what I do and look it up! The best resource for starting out is the Qiskit Textbook. Basically anything you want to do on paper you can do on Qiskit or at least give it a good try. Lets try a harder example together.



```
job=execute(qcircuit, M_simulator, shots=5000)
hist=job.result().get_counts()
print(hist) {'00': 1309, '01': 1319, '10': 1145, '11': 1227}
```

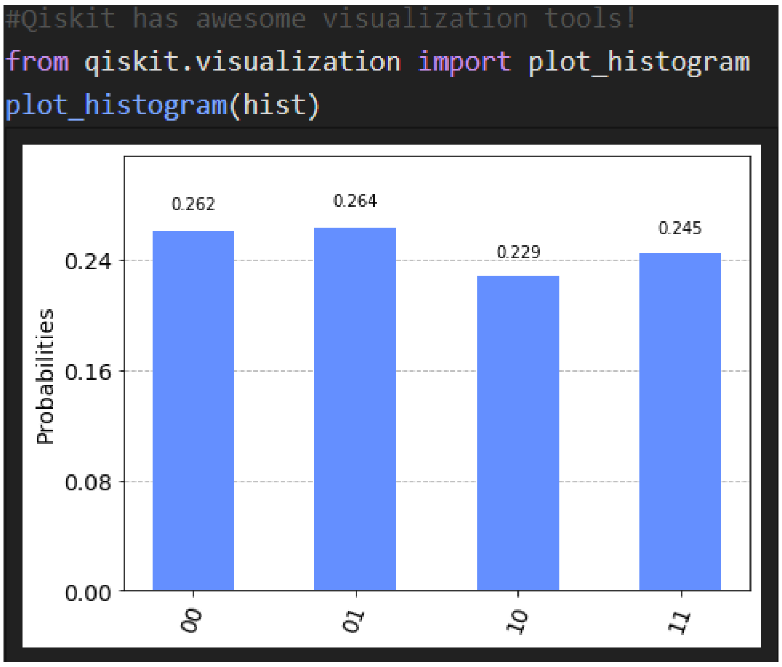
Figure 10: we get a dictionary key:value pair of our state and how many shots hit it

3 Quantum Teleportation

An example of a cool final project is going through one of the ciruits we did in class. The Quantum Teleportation protocol has alot of moving pieces and will be a good example to up our game. Please check out all the cool Quantum Protocols and Quantum Algorithims on the Qiskit site. They have all the ones we have covered in class included a few new ones. Teleportation works by having Alice wanting to send an arbitrary state of her qubit. $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

We want to send this same arbitrary state over to Bob's qubit. This is achieved by initializing



```
#Qiskit has awesome visualization tools!
from qiskit.visualization import plot_histogram
plot_histogram(hist)
```

Figure 11: really neat

Alice's qubit with some state $|\psi\rangle$, use a third ancillary state that will be entangled with Bob's qubit, and take measurements in the bell basis to determine the use of Z and X gates on Bob's qubit to make complete the teleportation.

We need to go back to the top of our circuit for the intialization of Alice's qubit to some state[5]. For example lets say $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. This is done by importing 'Initialzation' from qiskit.extensions and creating an initialization gate. We then append it to our zeroth qubit[6]

Finally, how do we check to see if any of this works? We first note that our init gate works by acting on Alice's qubit in the $|0\rangle$ and giving us some $|\psi\rangle$. Quantum computers unlike their jealous and classical cousins have the benefit of unitary operations that can be done by some other unitary operation. Qiskit lets us easily make an inverse init gate by figuring out what operation(s) undo our init gate. We do this because its easier to see if Bob's qubit is in the $|0\rangle$ state rather than the $|\psi\rangle$ state. Thats because $|\psi\rangle$ can be in a superposition of states meaning we could not get a certain measuremnt.

Now that this is all complete we can use '.draw(output='mpl')' to get a really nice picture again. I am really proud of this one. You should be too, this is not trivial stuff.

Bob's qubit was the second qubit[7], Qiskit unfortunately prints states backwords! So some state $|abc\rangle$ gets written as $|cba\rangle$ so Bob's qubit will be the leftmost one in our results coming up. What we are looking for after we measure the circuit is Bob's qubit always being in the $|0\rangle$ state. So we use our M simulator just as we did earlier.

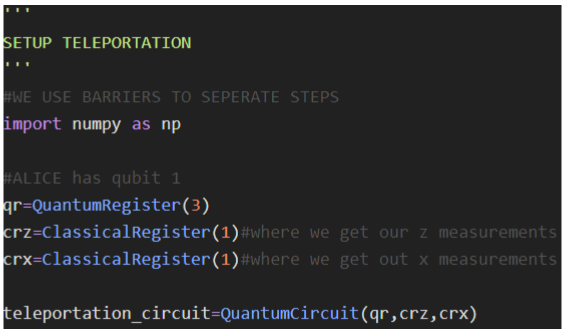
Check that out! The states we measure are $|000\rangle$, $|100\rangle$, $|010\rangle$, and $|110\rangle$. In all possibilities we have our last qubit in the $|0\rangle$ state! Thats a success and you have coded a quantum algorithm, way to go[8].

[5]Python much like quantum circuit reads instructions one at a time starting at the first instruction and ending at the last, we need to write our code in a desired order to get a desired result

[6]this is also the first qubit and known as Alice's qubit. A qubit by any other name is just as random.

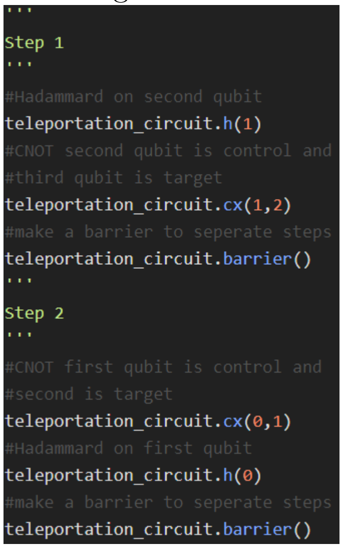
[7]this is also the third qubit or Bob's qubit, no one likes sematincs

[8]I'm proud



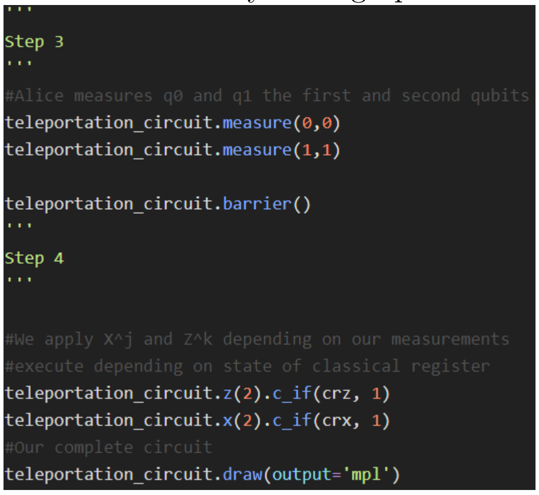
```
'''
SETUP TELEPORTATION
'''
#WE USE BARRIERS TO SEPERATE STEPS
import numpy as np

#ALICE has qubit 1
qr=QuantumRegister(3)
crz=ClassicalRegister(1)#where we get our z measurements
crx=ClassicalRegister(1)#where we get out x measurements

teleportation_circuit=QuantumCircuit(qr,crz,crx)
```

Figure 12: tweaking old code for teleportation



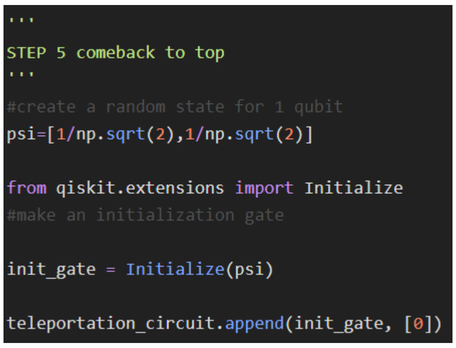
```
'''
Step 1
'''
#Hadammard on second qubit
teleportation_circuit.h(1)
#CNOT second qubit is control and
#third qubit is target
teleportation_circuit.cx(1,2)
#make a barrier to seperate steps
teleportation_circuit.barrier()
'''
Step 2
'''
#CNOT first qubit is control and
#second is target
teleportation_circuit.cx(0,1)
#Hadammard on first qubit
teleportation_circuit.h(0)
#make a barrier to seperate steps
teleportation_circuit.barrier()
```

Figure 13: We start by setting up the circuit...



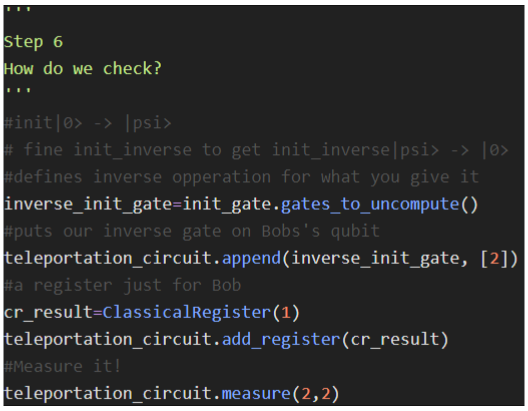
```
'''
Step 3
'''
#Alice measures q0 and q1 the first and second qubits
teleportation_circuit.measure(0,0)
teleportation_circuit.measure(1,1)

teleportation_circuit.barrier()
'''
Step 4
'''

#We apply X^j and Z^k depending on our measurements
#execute depending on state of classical register
teleportation_circuit.z(2).c_if(crz, 1)
teleportation_circuit.x(2).c_if(crx, 1)
#Our complete circuit
teleportation_circuit.draw(output='mpl')
```

Figure 14: ...and finish it with measurements



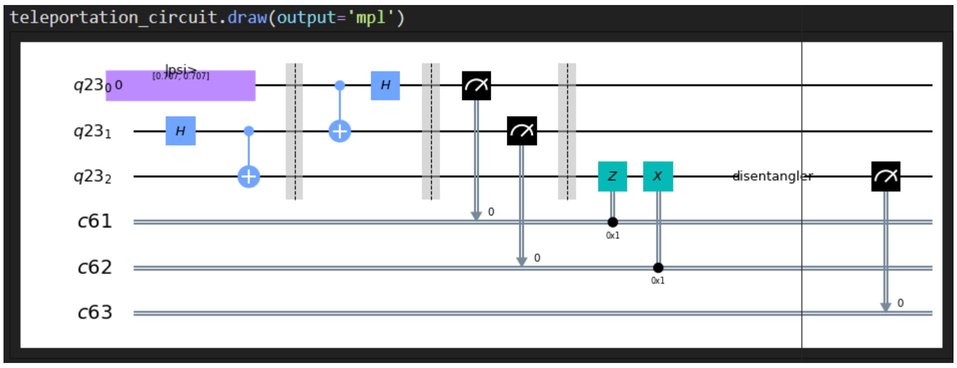
```
'''
STEP 5 comeback to top
'''
#create a random state for 1 qubit
psi=[1/np.sqrt(2),1/np.sqrt(2)]

from qiskit.extensions import Initialize
#make an initialization gate

init_gate = Initialize(psi)

teleportation_circuit.append(init_gate, [0])
```

Figure 15: Some adventures can only end by going back to the beginning



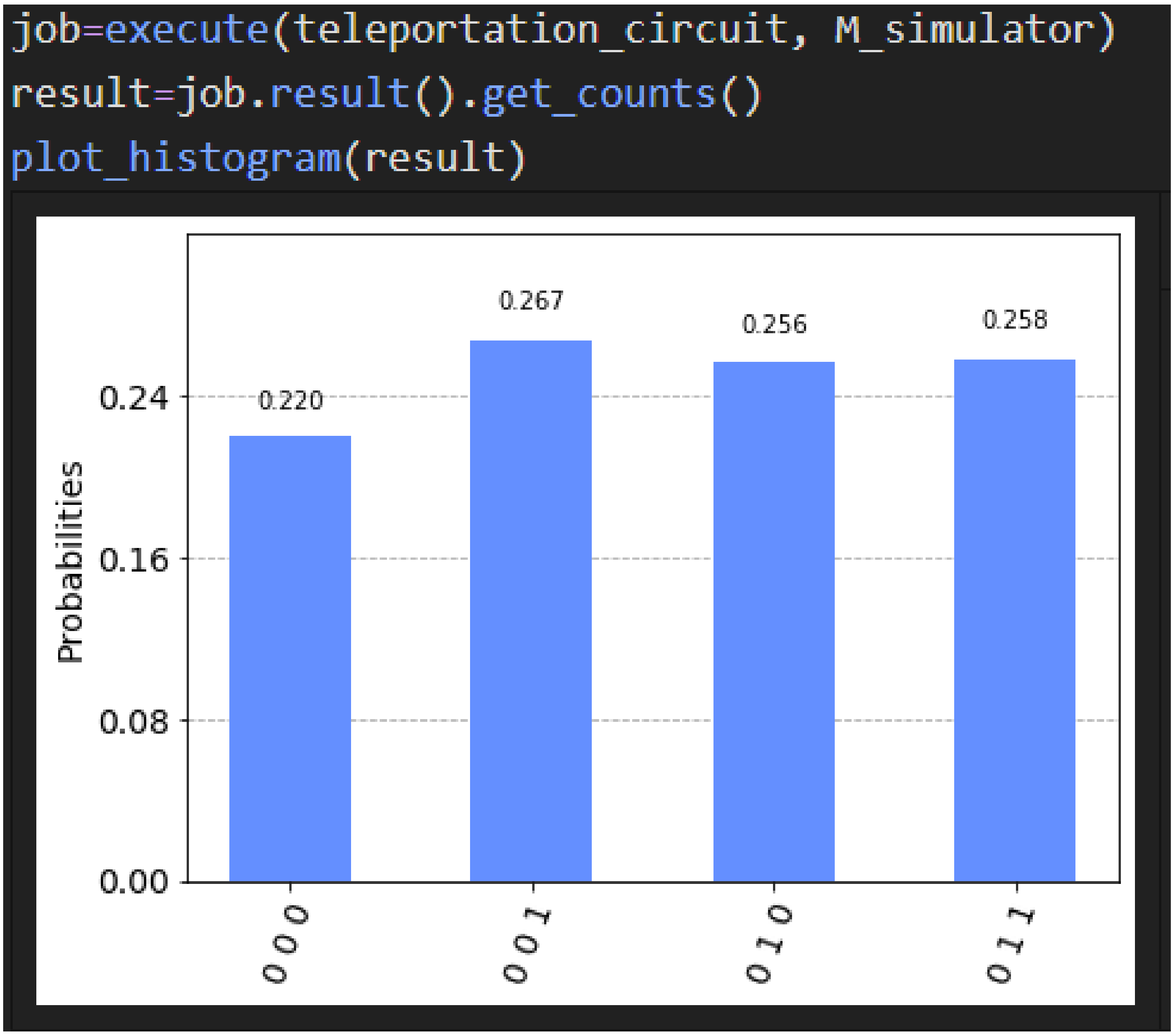
```
'''
Step 6
How do we check?
'''
#init|0> -> |psi>
# fine init_inverse to get init_inverse|psi> -> |0>
#defines inverse opperation for what you give it
inverse_init_gate=init_gate.gates_to_uncompute()
#puts our inverse gate on Bobs's qubit
teleportation_circuit.append(inverse_init_gate, [2])
#a register just for Bob
cr_result=ClassicalRegister(1)
teleportation_circuit.add_register(cr_result)
#Measure it!
teleportation_circuit.measure(2,2)
```

Figure 16: Create an Inverse gate and lets measure Bob's qubit.



```
teleportation_circuit.draw(output='mpl')
```

Figure 17: I dont know why disentangler shows up but it basically disentangles as far as I can tell



```
job=execute(teleportation_circuit, M_simulator)
result=job.result().get_counts()
plot_histogram(result)
```

Figure 18: Results!