How It Works and Overhead Analysis

A code that is easy to code for the programmer is difficult to execute by the computer.



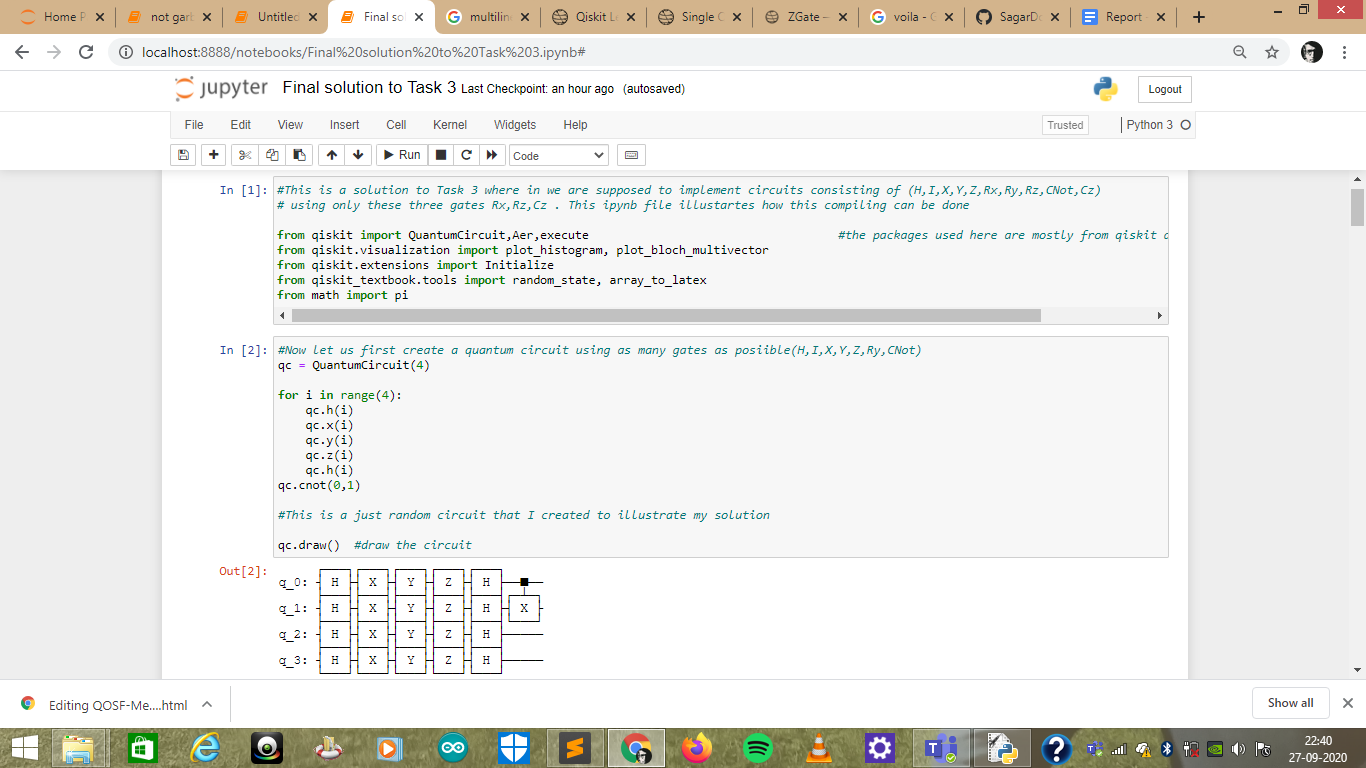
**Sagar B Dollin**

04.02.1998

sagardollin@gmail.com

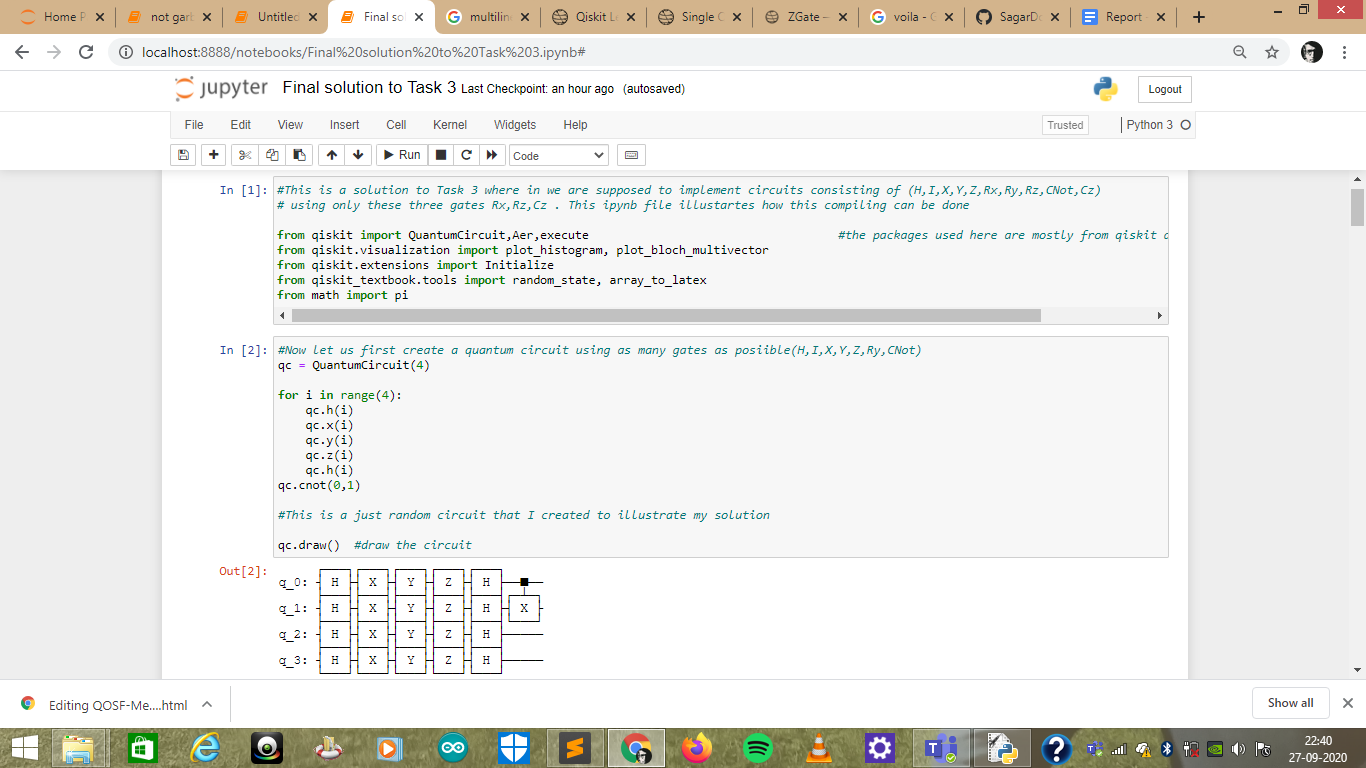
# How The Solution Works?

It's quite easy to understand. First of all let's see what are the different packages we are using.

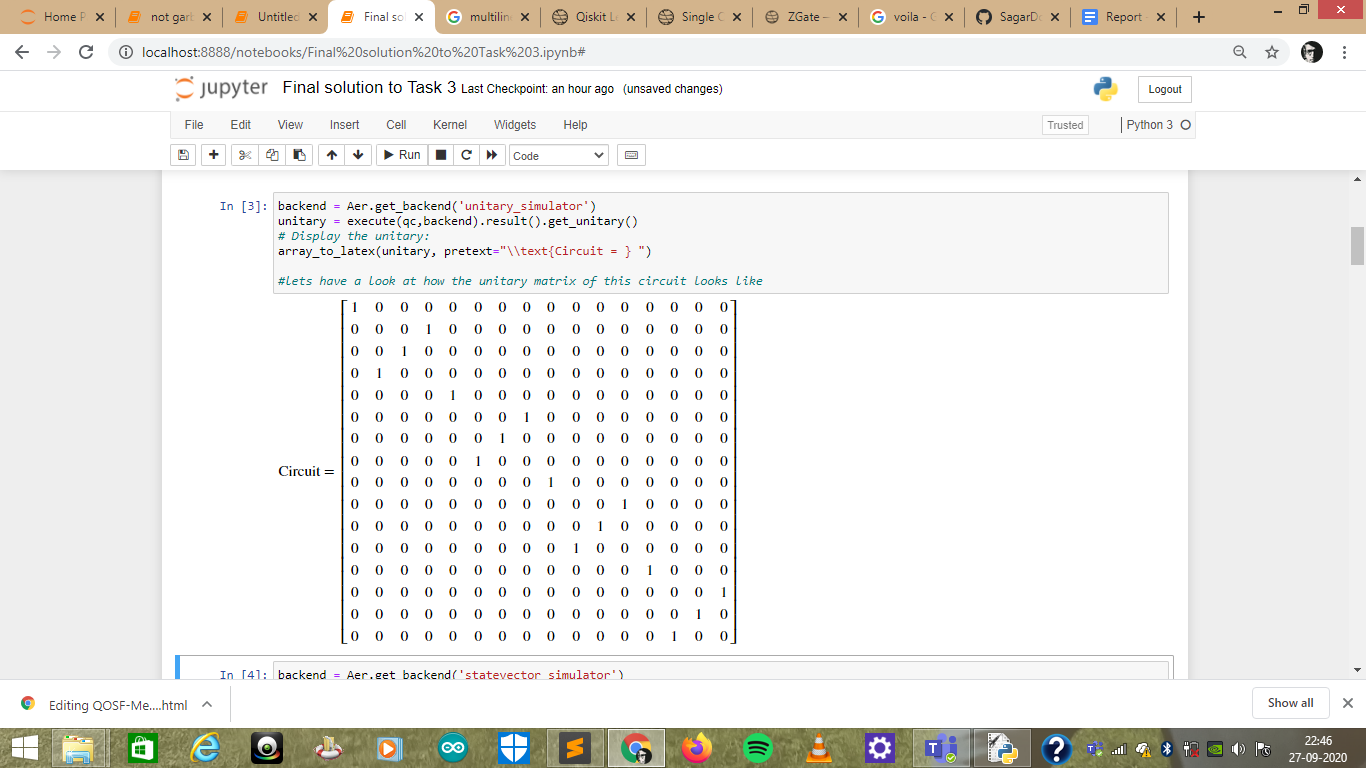


I installed these when I was studying Qiskit Textbook. So I'm assuming the reader is pretty much familiar with the Qiskit and Qiskit Textbook Packages.

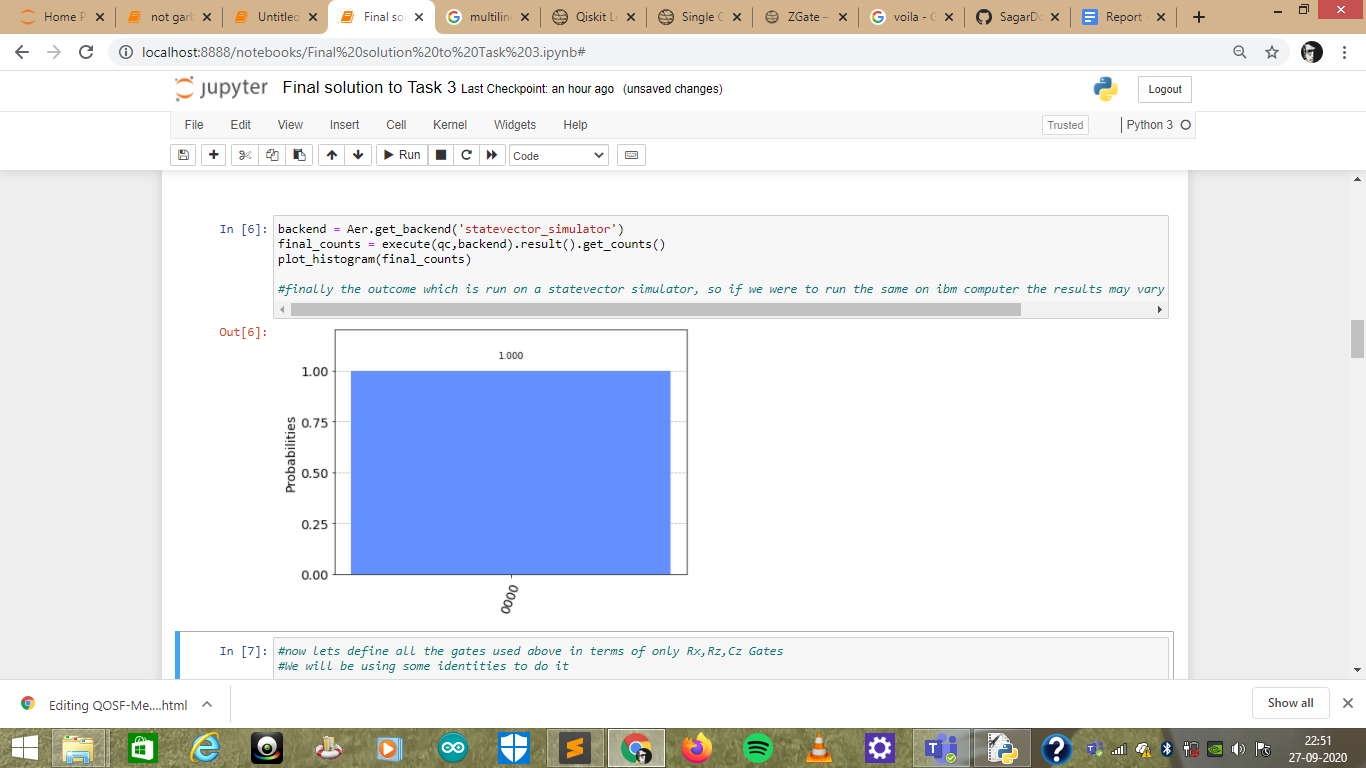
Next, Let us implement a quantum circuit of say 4 qubits, as shown below



Let’s see the unitary of the above Quantum circuit.

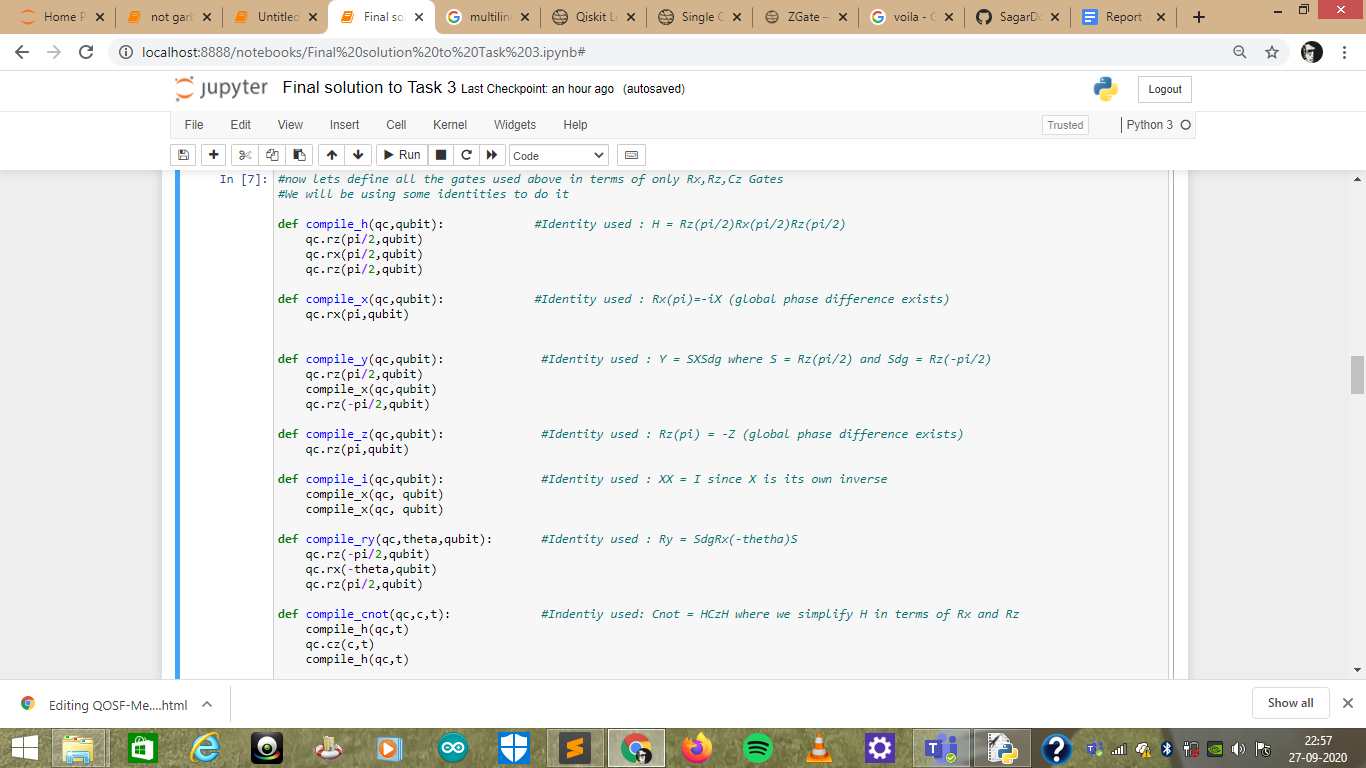


It's beautiful, isn't it? That is because it looks like a 16x16 Identity Matrix. This is the point where I realised XYZ=I. Now since our Quantum circuit has a unitary of the Identity Matrix we can be sure that the state of the system doesn't change, i.e., we begane with |0000> state and after measuring the circuit at the end we will get a result of |0000>. Lets visualise the result

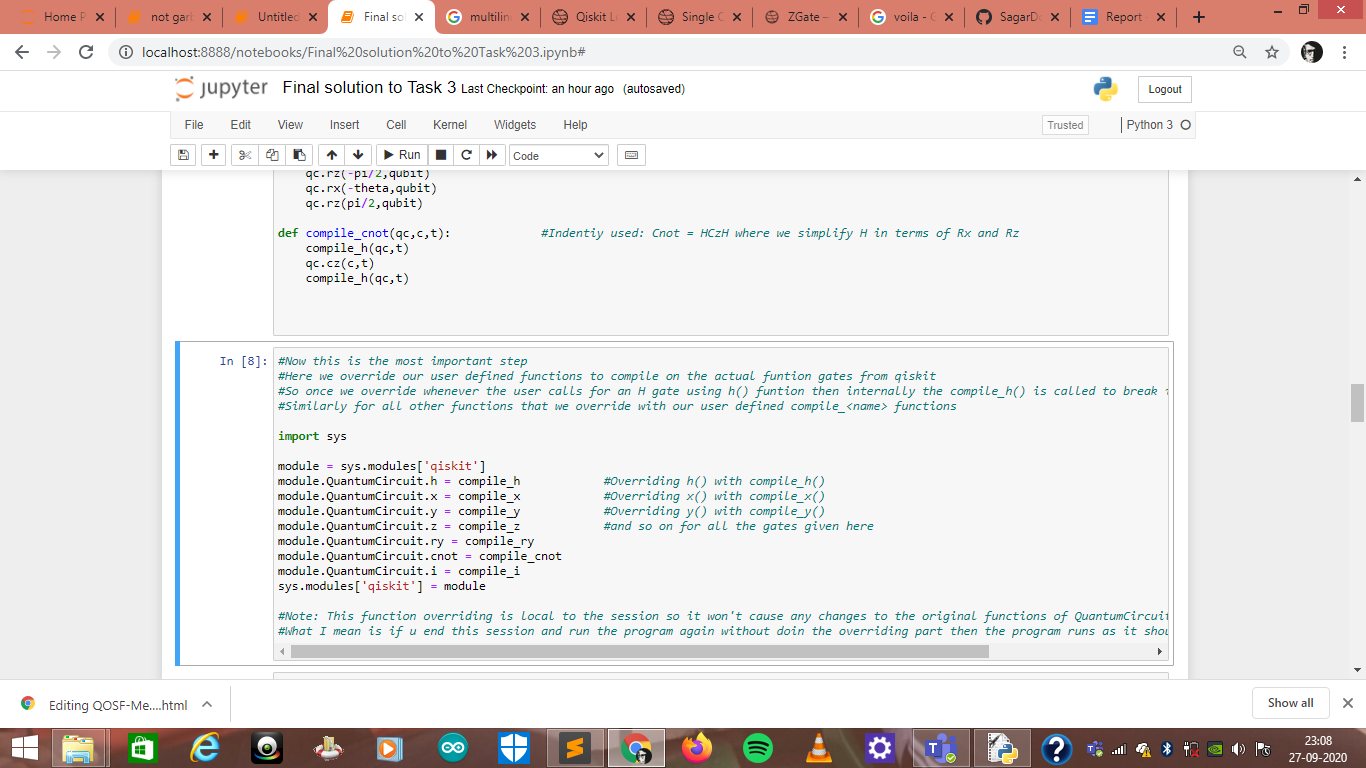


Now keeping in mind the result of our Quantum circuit let’s design a logic that can convert this Circuit or perhaps any Quantum circuit(H,I,X,Y,Z,Rx,Ry,Rz,Cnot,Cz) into a Circuit comprising of only three basic gates i.e., Rx,Rz,Cz. The idea is to define functions to compile each gate into its equivalent form defined by using only Rx,Rz and Cz gates using some identities. These identities can change the global phase but they should yield the same result as yielded by the original Quantum Circuit.

Have look at the below code defining functions to convert each gate to its equivalent form ignoring the global phase difference

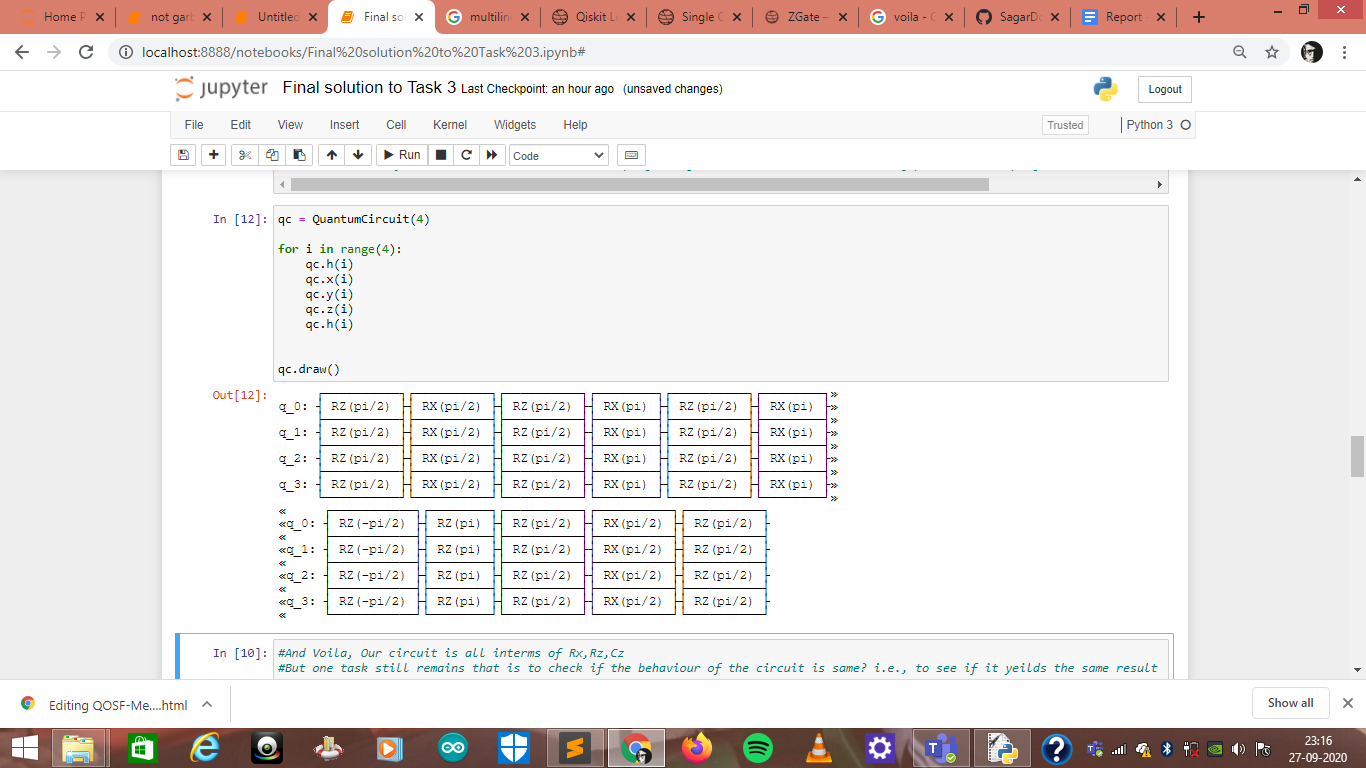


Now we **override** the default methods of the gates from Qiskit Quantum Circuit with the above user defined functions that we defined.



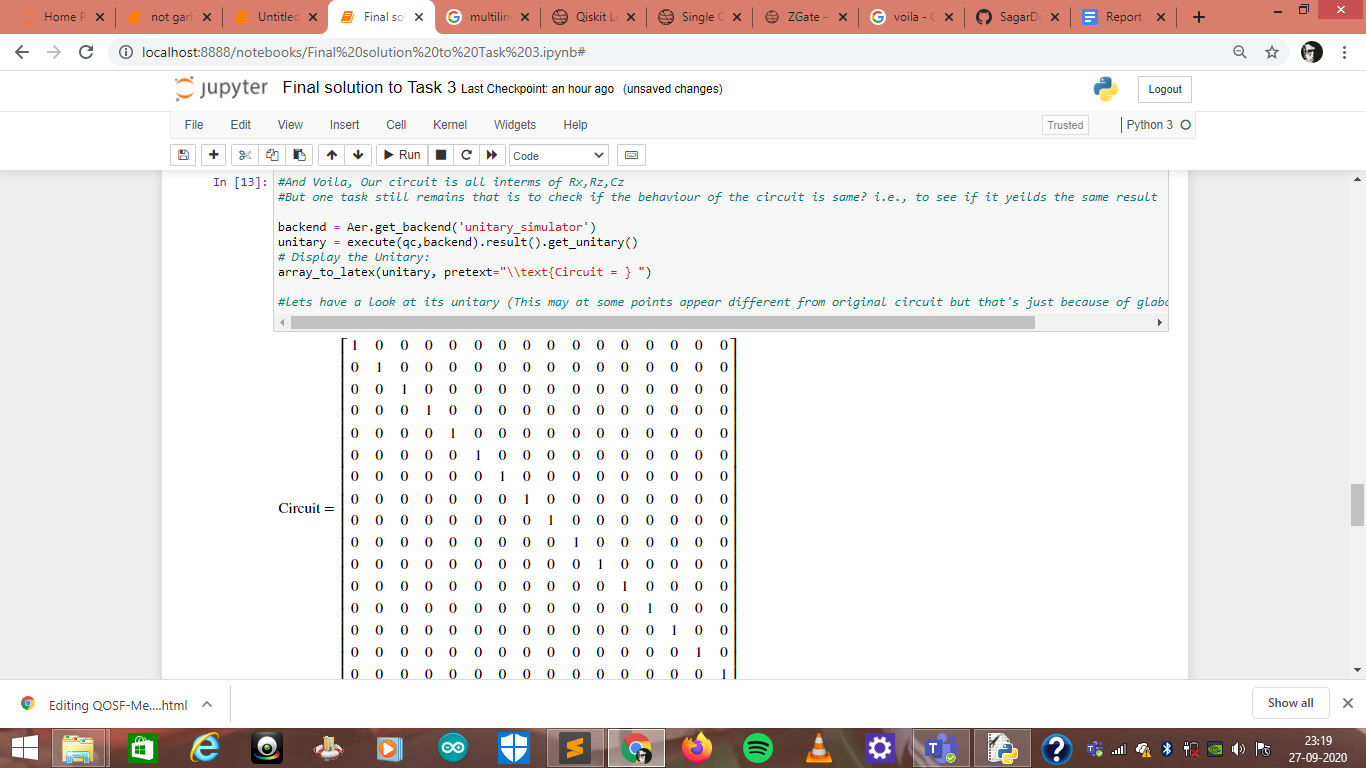
What we have done in the above code is we are overriding the h() function that is used to apply hadamard gate by our user defined function compile\_h(). This compile\_h() function gives the same result as h() but it is implemented using Rx and Rz gates. So once we run this code , from then onwards whenever we call h() method the compile\_h() is called and h is implemented using these. The same is true for all the methods we are overriding here. Note that this overriding is local to the session therefore it will not cause any changes to the qiskit’s original default methods.

Now, after overriding let's implement the same quantum circuit that we did in the beginning, as shown



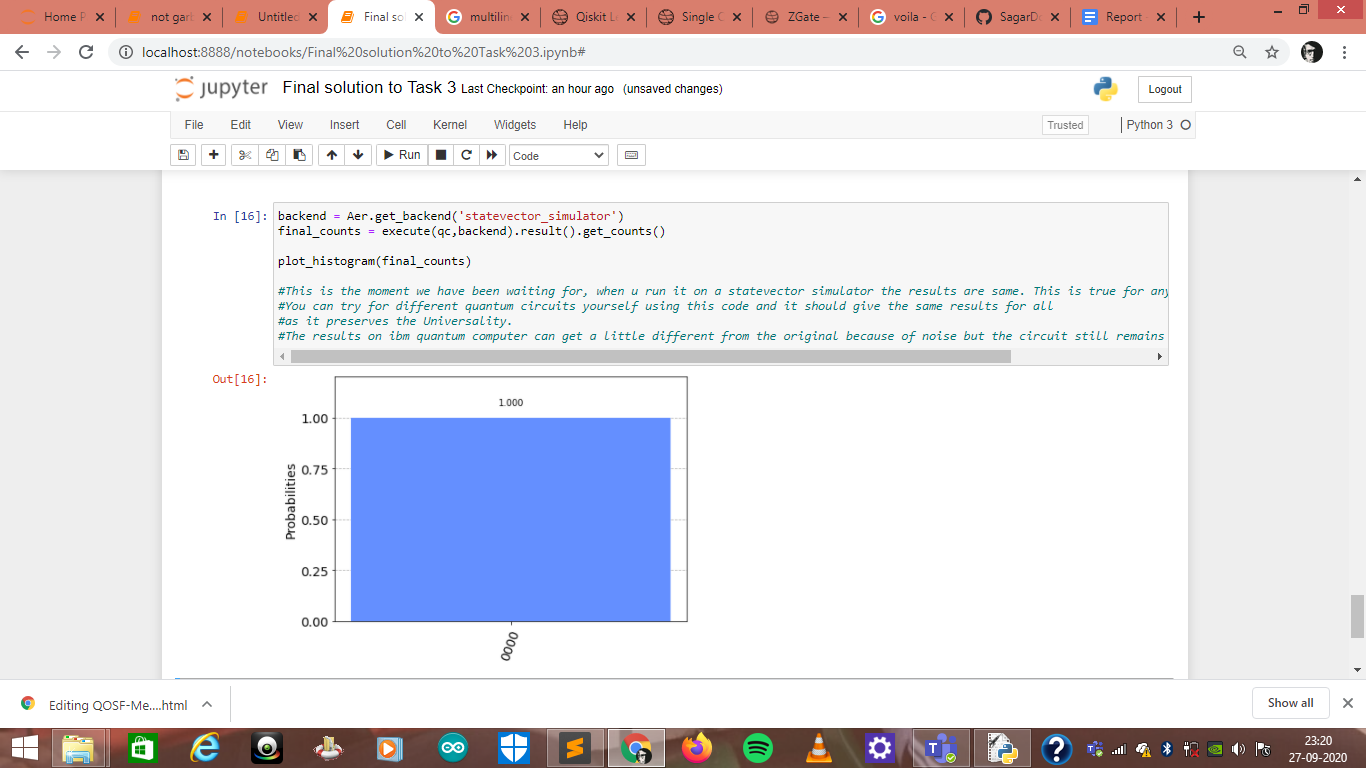
This gives us all the gates defined only in terms of what we wanted i.e., as per the requirement of the Task.

Now to check if the results are correct lets have a look at the Unitary matrix and the state vector probability



The Unitary is the same, although in this example we don't have any global phase difference but if we take different examples we will come across examples where we will encounter global phase difference.

Let's have a look at the State Vector probability



And allow me to say “YIPEEEEEEEEE!!”

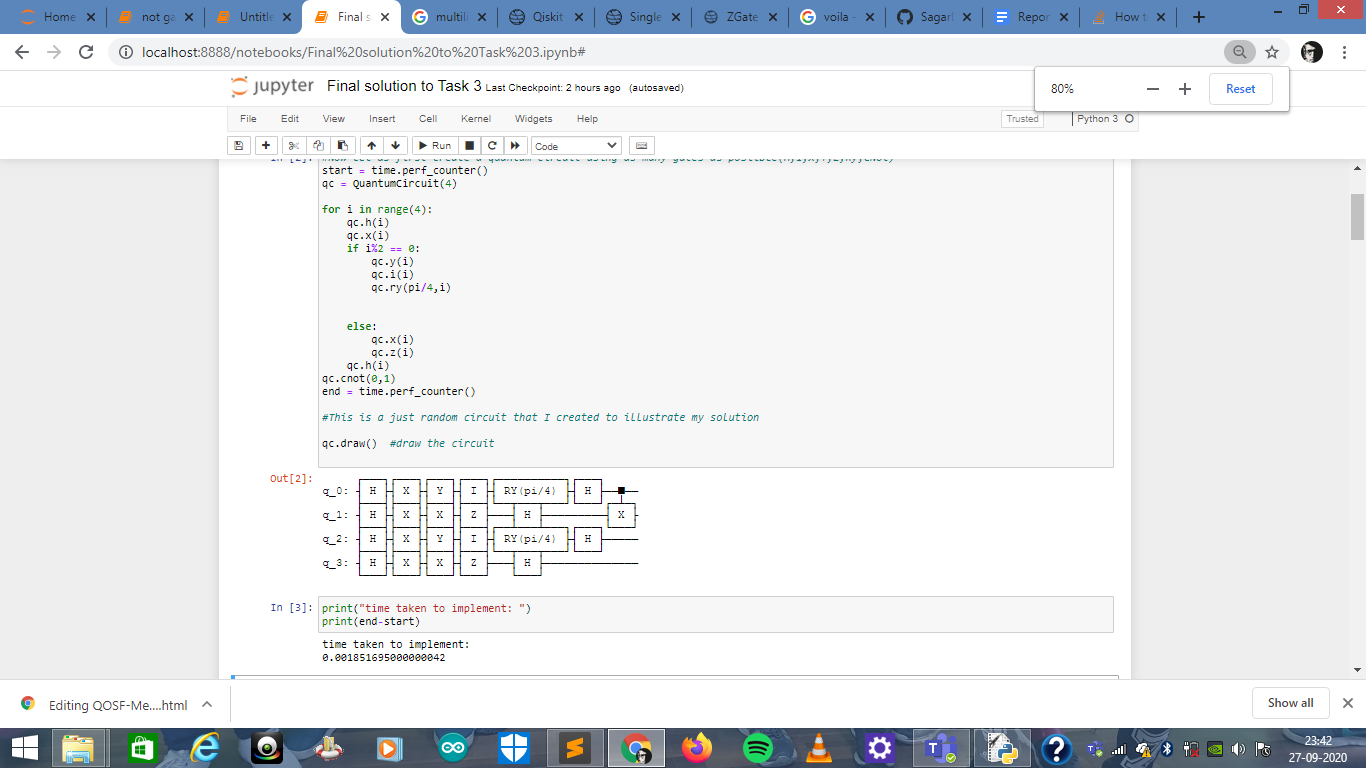
But our task here is not yet done let's dig into the overhead analysis

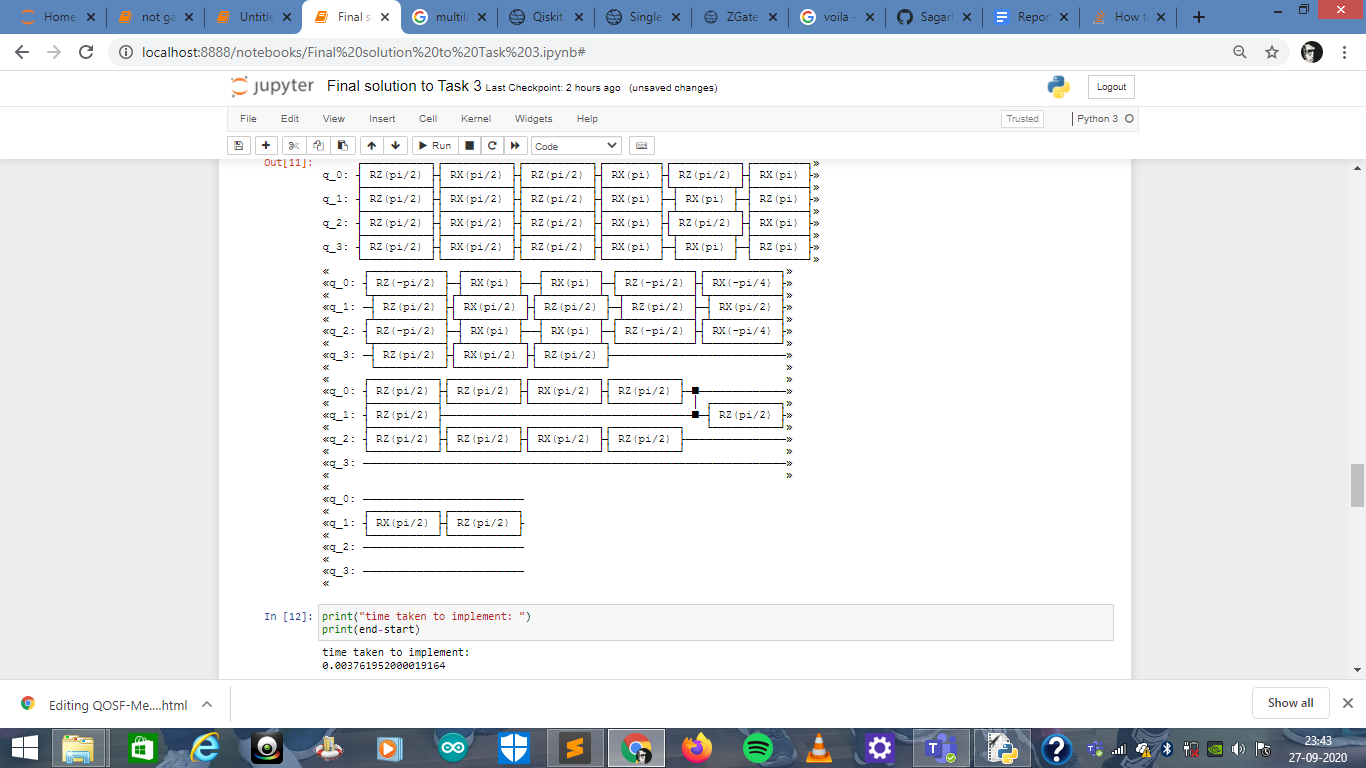
# Overhead Analysis

In my understanding I’m considering these things as the overhead

1. Time Complexity
2. Number of gates or the depth or the circuits

**Time Complexity**  
In my solution every gate is converted to its equivalent in some constant time, so i can say that if there are n gates in the circuit my implementation takes O(n) to implement the gates , which is the same as without my logic so implementation doesn't add any overhead in asymptotic notations but let's see quantitatively what is the difference in time for both(this could be machine dependent)





As we can see it takes almost double the time required for implementing the circuit in terms of rx,rz,cz , also notice I have used a more complex circuit than the earlier example. Double time means O(2n) which is equivalent to O(n) which is still a linear solution to the problem.

**Number of Gates(Depth of the Circuit)**

The number of gates implemented in the circuit can be given by the formula:

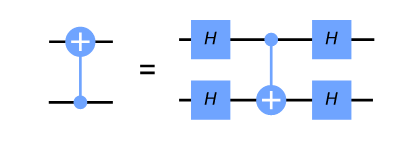
***Total number of gates = 2h+2y+i+2ry+6cnot+original\_gates***

Where h is number of hadamard gate in original input circuit.

Similarly y,i,ry,and cnot are all numbers of corresponding gates in the original input gate and Original\_gates is the total number of gates in the original input gates.

As we can see this increase in the number is not optimal. Can we do better? And the answer is yes of course we can, let me discuss some approaches that can be used to optimize our solution in terms of the number of gates used.

First we need to keep track of **phase kick back**, for example see this image

source: Qiskit textbook

Whenever we encounter an input like the one in the left side we can just implement the gate on the right side, so instead of using 5 gates we use only one of them. This can be achieved using variables or lists to keep track of the structure of the circuit and replace it with appropriate optimised circuits.

Next approach is a more generalised form of what we saw above , we use identities to identify circuits that are equivalent to a smaller circuits for example

If the Input is HZH then by the identity we know that

HZH = X

And X can be implemented using Rx(pi) so this way we achieve our output circuit more compact than the input circuit.

All these can be achieved using more advanced algorithms by keeping track of the input gates using appropriate data structures.

Another interesting fact to note is we are implementing cnot gate using cz gate, but in reality on ibm quantum computer cnot is the only 2 qubit gate that can be implemented and all other 2-qubit gates are implemented using some combination of cnot and other gates. So if we run our solution on a ibm quantum computer, whenever we encounter a cnot in our input circuit we are implementing cz to achieve cnot, we use 7 gates to do this and again ibm quantum computer uses another 3 gates to apply cz that we produced to implement in terms of cnot.