

Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Telecommunications and Media Informatics

**Project Lab Report**

**Title: Quantum Computing**

**Written by:** Yagublu Nurlan, Mirzayev Huseyn **Neptun#:** G72RIJ, SYWTRE

**Field:** Computer Science Engineering

**Specialization:** Internet Architecture and Services

**E-mail:** [**nurlanyagublu@gmail.com**](mailto:nurlanyagublu@gmail.com)**,** [**hsynmrz28@gmail.com**](mailto:hsynmrz28@gmail.com)

**Supervisor: Prof. Imre Sándor**

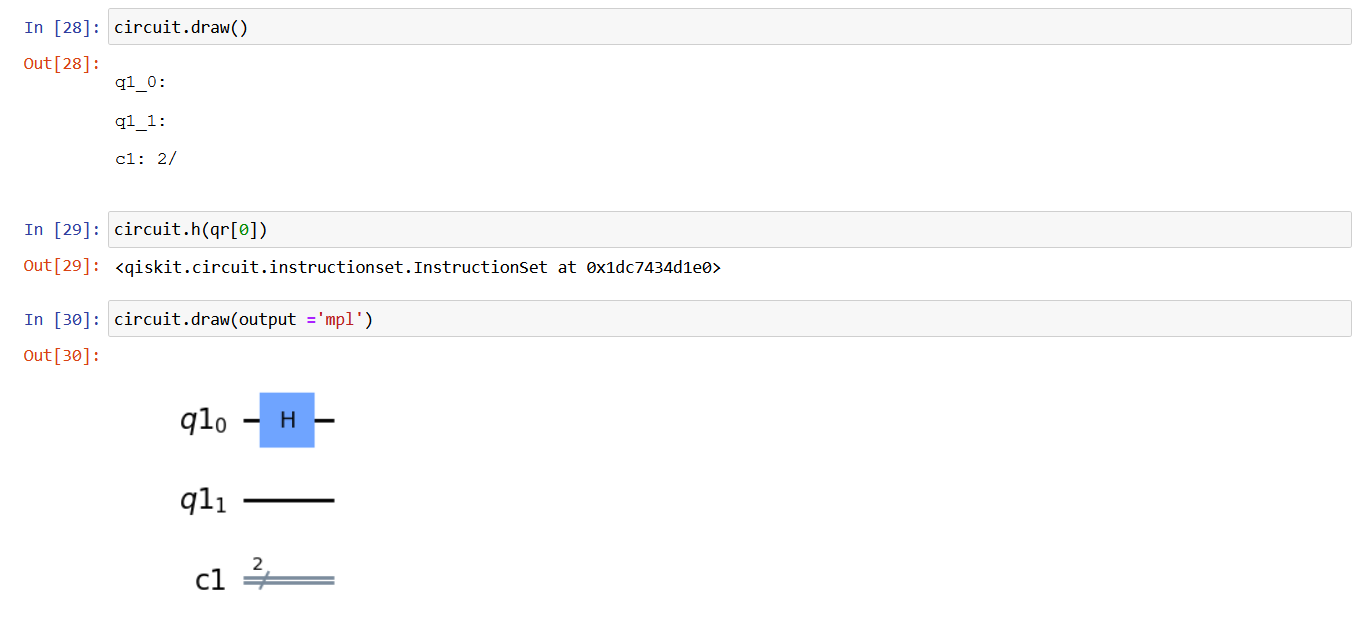
**E-mail:** **imre@hit.bme.hu**

**Academic year:** 2022/2023 Autumn semester

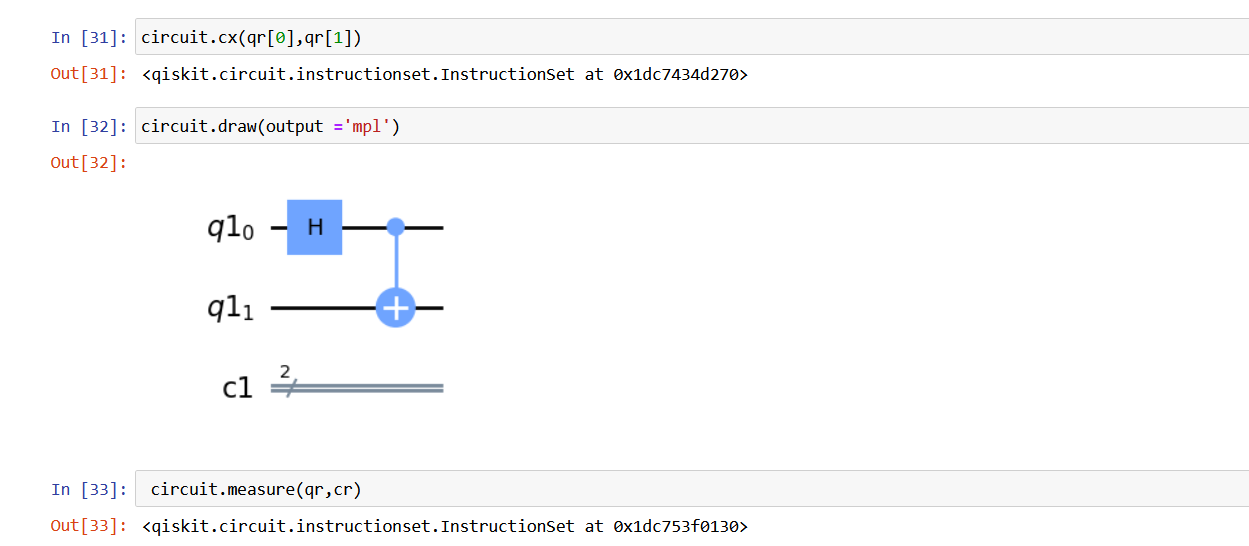
**SIMPLE PROBLEM**



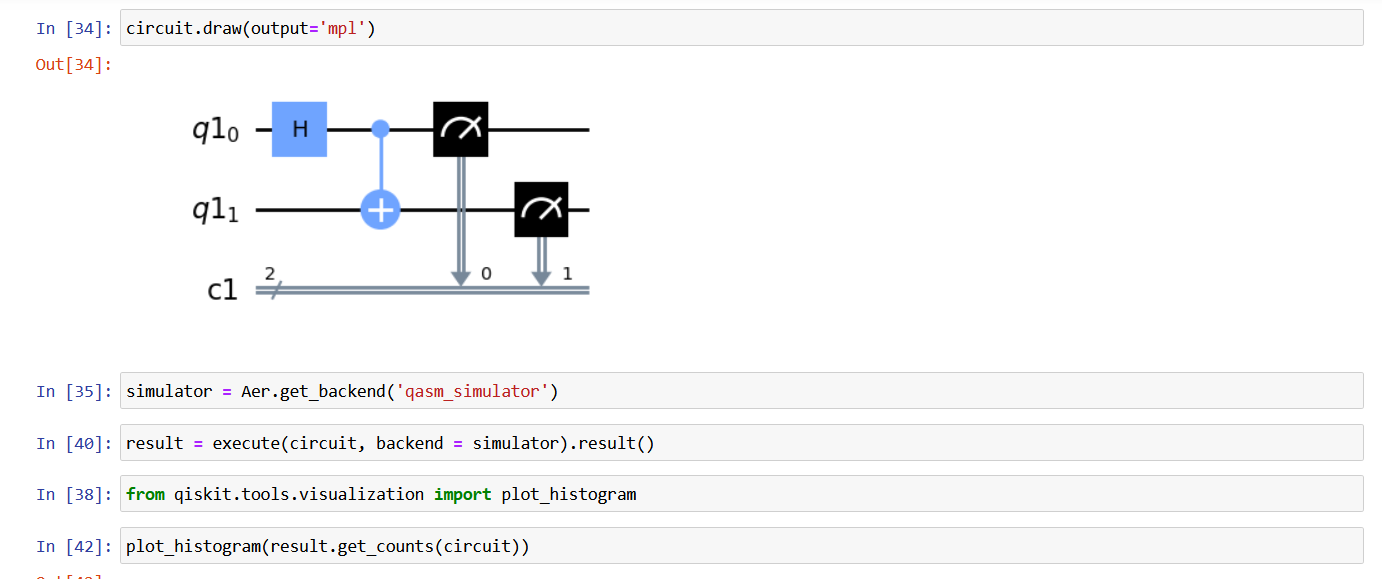
We imported the important libraries and created two quantum bits and two classical bits for their representation.



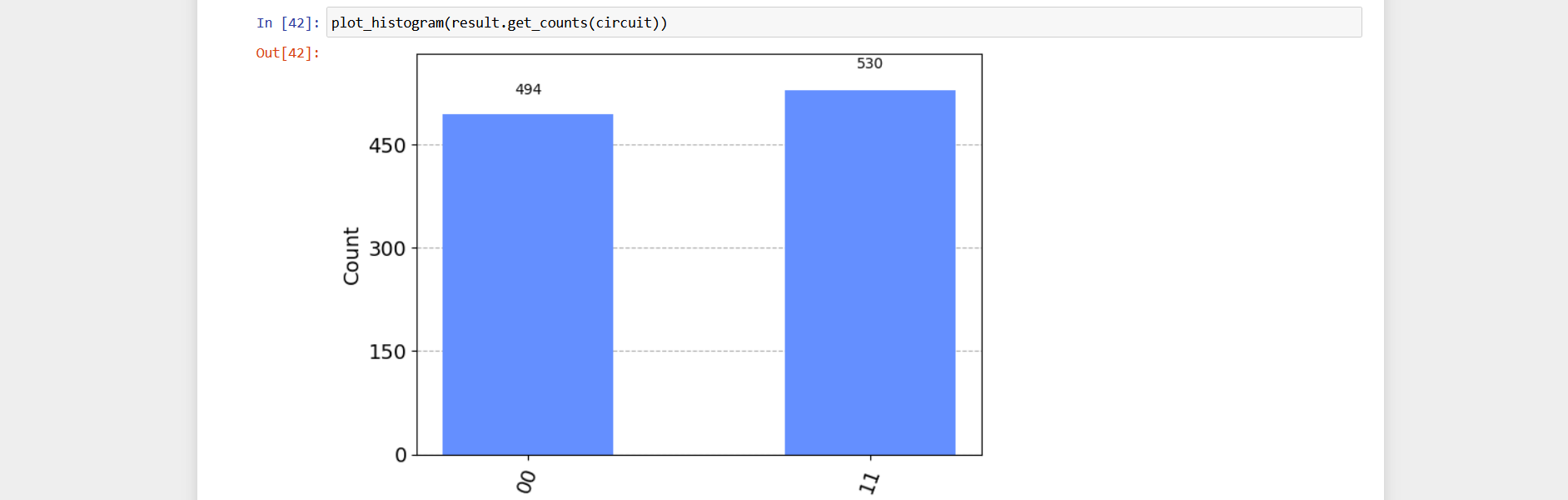
We then used the matplotlib to visualize our circuit and added Hadamard gate to first quantum bit.



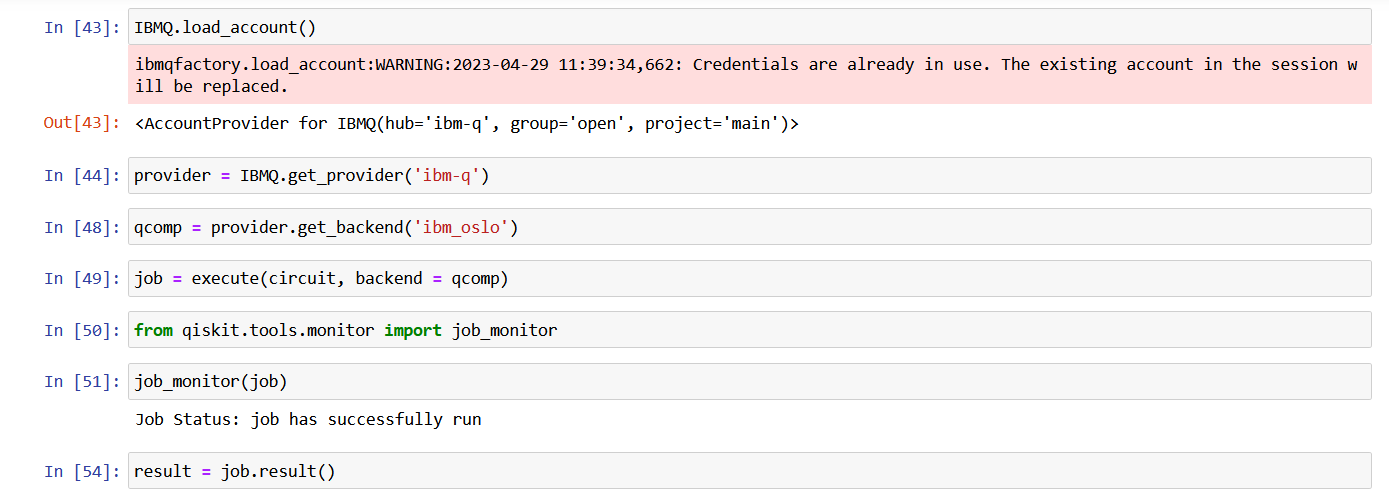
We then proceeded to applying the CX gate to the circuit and have measured it.



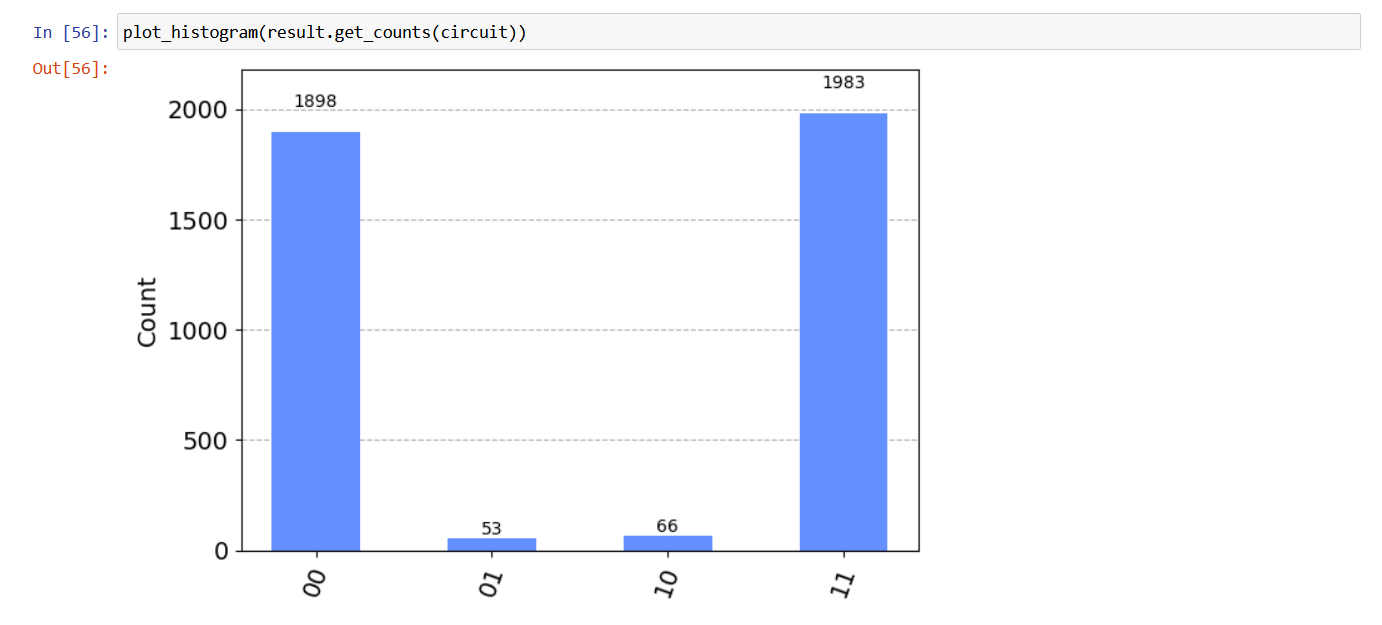
The circuit now looks like in the picture above and we need to run and initialize it. For that task we will use the local quantum simulator that we can get from the Aer library. We import another library called plot\_histogram to visualize all the results as a chart. By executing the circuit using the QASM simulator we store the result and then visualize it with the Plot Histogram:



This is a representation of the results using a local quantum engine, but now we will use the IBM cloud computers to run our code and we will do the next steps in order to achieve our goal:



We load the credentials to the IBM account and get the backend from that said cloud. We save our code that uses cloud simulator into a variable and pass that variable to a queue that will be executed by the machine in some time.



We can use Plot Histogram again to view our result and this time all the possible outcomes are tested and test count is much higher than on the local machine. This was a simple task to implement our Qiskit knowledge into work and we will move on to solving algorithms in Qiskit.

**DEUTSCH JOZSA ALGORITHM**

The Deutsch-Jozsa algorithm is a quantum algorithm that determines whether a given Boolean function is balanced (returns an equal number of 0's and 1's for different inputs) or constant (returns the same output for all inputs).

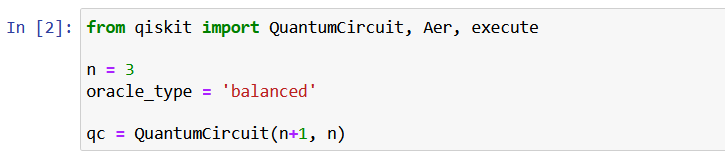
Classically, this problem requires evaluating the Boolean function for all possible inputs, which takes O(2^n) time, where n is the number of input bits. However, the Deutsch-Jozsa algorithm can solve this problem in just one evaluation of the Boolean function using a quantum computer.

The algorithm involves preparing an n-qubit input state in a superposition of all possible inputs, applying a quantum oracle that implements the given Boolean function, and then applying a series of Hadamard gates and measurements to the input qubits. The resulting measurement probabilities will be different for a constant function and a balanced function, allowing us to determine the type of function using just one evaluation.

The Deutsch-Jozsa algorithm was one of the first quantum algorithms discovered and provides a simple example of how quantum computers can outperform classical computers for certain problems.

We can implement the Deutsch-Jozsa algorithm in Jupyter Notebook just as follows:

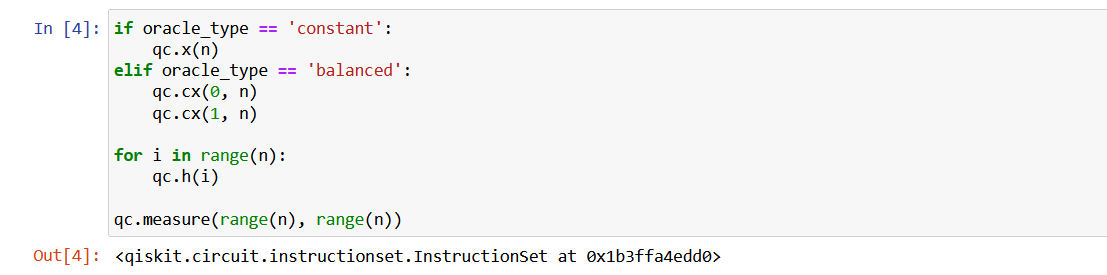
We import necessary libraries and modules and define the number of qubits and the type of oracle(3 Qbits and balanced).



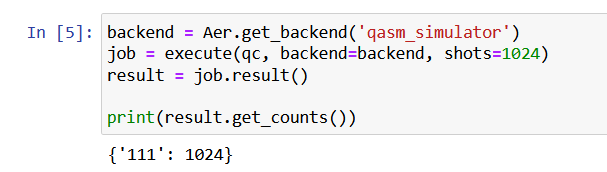
Then proceed to initialize the input qubits to |0> and apply Hadamard gates to all input qubits:



We then apply the oracle function and apply Hadamard gates to all input qubits. Then measure all input qubits:



And finally we simulate the circuit and get the results:



**GROVER'S ALGORITHM**

Grover's algorithm, which was developed by Lov Grover in 1996, is a well-known quantum algorithm. Initially designed for unstructured search problems, where the goal is to find a specific item in an unorganized database, Grover's algorithm has since become a subroutine for various other algorithms, including Grover Adaptive Search.

The Grover class in Qiskit is responsible for implementing Grover's algorithm, along with its generalized version called Amplitude Amplification. This class provides the flexibility to adjust parameters such as the number of iterations and other meta-settings for Grover's algorithm.

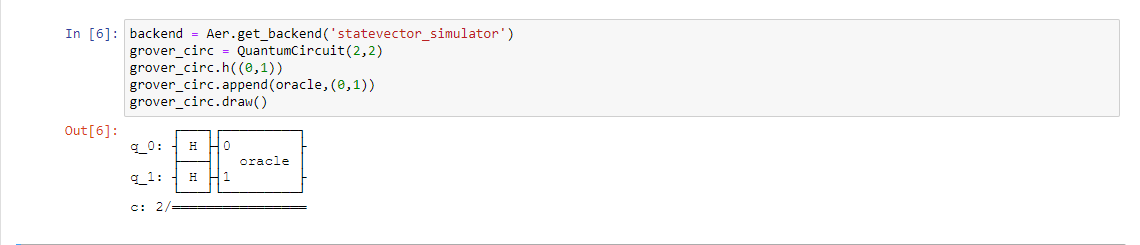
We have an unsorted list of N elements, and we want to locate a particular element of the list. In the classical case, the complexity of this problem scales as order N. In quantum computers we can handle only square root N calls to the Oracle by using Grover’s algorithm.

To show how this algorithm works we can encode our inputs as the basis state of a quantum computer. For 2 Qbits we can represent 4 input states: 00, 01, 10, 11. Suppose that our winner is the state 11. So if we feed our state 11 into our Oracle, where we get back -11 and this is the way oracle is gonna work. It’s gonna flip the sign of the input if the input is the winner. The way we can change these states is through a unitary matrix which we can encode into a quantum circuit. So we want a circuit that flips the sign of the 11 state and conveniently we have a gate that does just controlled-Z gate: |11> — [CZ] — -|11>. We also need a Reflection operator in order to do amplitude amplification. The combination of the Oracle and Reflection operator is known as Grover diffusion operator.

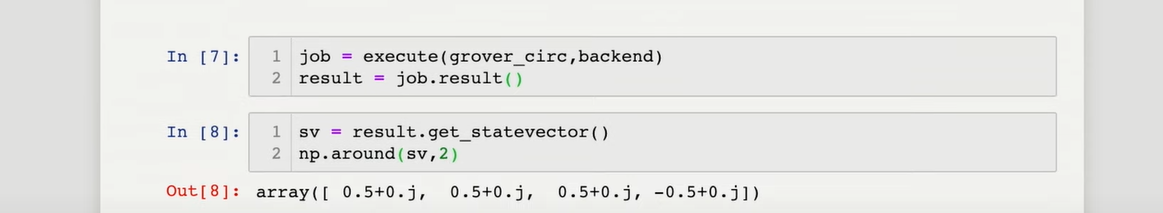
We created CZ gate which acts as Oracle:



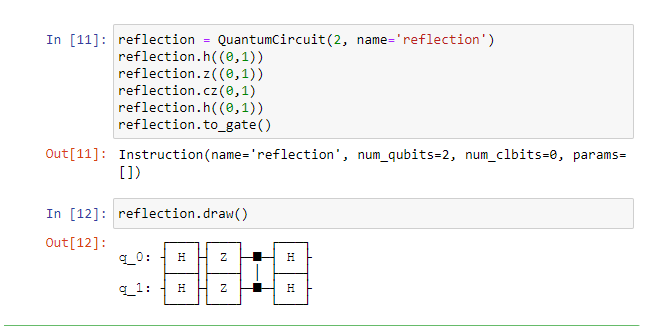
To check oracle we prepared a superposition state of all qubits by applying a Hadamard gate on each one of them. We called the state s for Superposition:



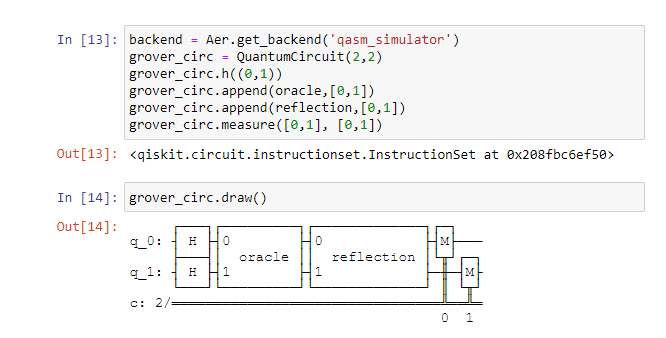
We executed this job on the simulator:



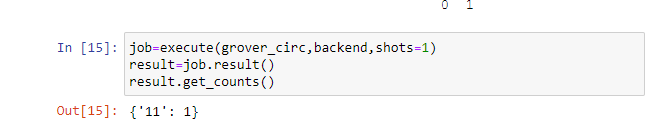
But we are not done yet, we have to square the state factor in order to get back probabilities of measuring those states. We used a reflection operator for being able to do amplitude amplification. With amplitude amplification we can amplify the probabilities of the winning state w and we can reduce the probabilities of all the non-winning states.

We have added reflection operator:  


Afterwards, we put it all together(using a qasm simulator) and prepared our superposition state, now we have our oracle and reflection operator.



At the end we got the state 11. Here is the result:



**PHASE ESTIMATION**

Phase estimation is an important algorithm in quantum computing that allows us to estimate the phase of an eigenvalue of a unitary operator. In quantum computing, unitary operators are represented by quantum gates that operate on qubits. The phase estimation algorithm aims to determine the phase factor associated with a specific eigenvalue of a unitary operator.

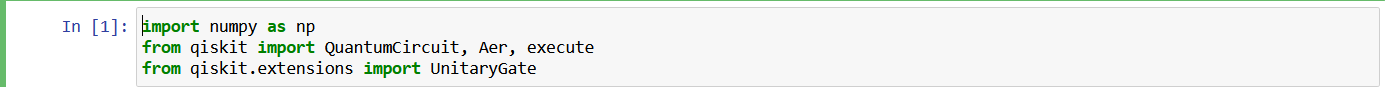
The algorithm involves the following steps:

1. Prepare an input state: This register is called the "control qubits."
2. Apply the unitary operator: This operation entangles the control and target qubits.
3. Apply the inverse quantum Fourier transform (QFT): The QFT is a quantum algorithm that converts the information in the amplitudes of a quantum state into their corresponding Fourier coefficients.
4. Measurement: Measure the control qubits.
5. Post-processing: This step might involve additional classical computations

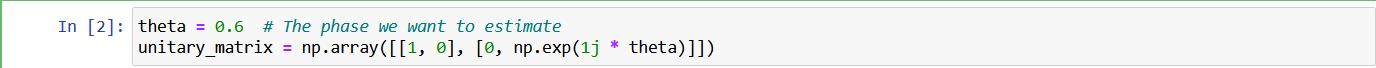
By performing this algorithm, we can estimate the phase of the eigenvalue with increasing accuracy as the number of control qubits increases.

Here's an example of phase estimation using Qiskit in a Jupyter Notebook. This example demonstrates how to estimate the phase of a given unitary operator using the Quantum Phase Estimation (QPE) algorithm.

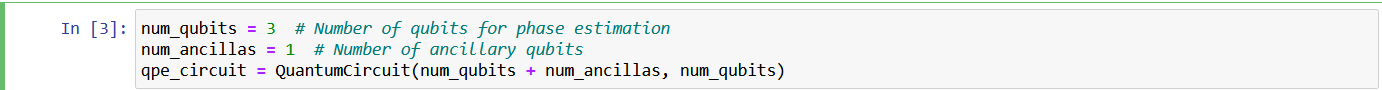
1. Import necessary libraries.



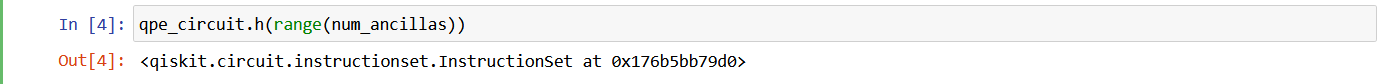
1. Define the unitary operator whose phase we want to estimate:



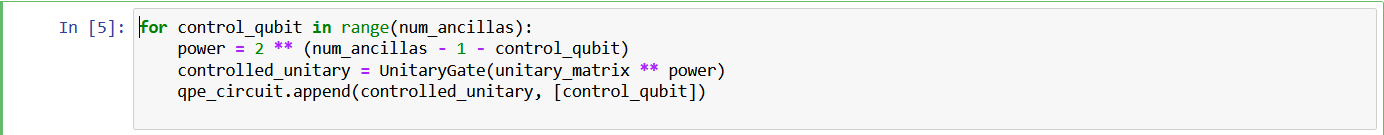
1. Define the quantum circuit for phase estimation:



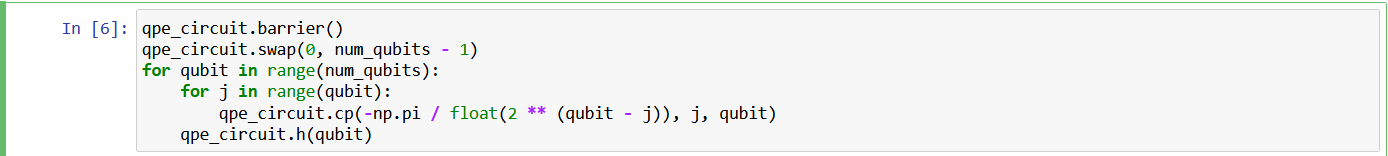
1. Apply Hadamard gates to ancillary qubits:



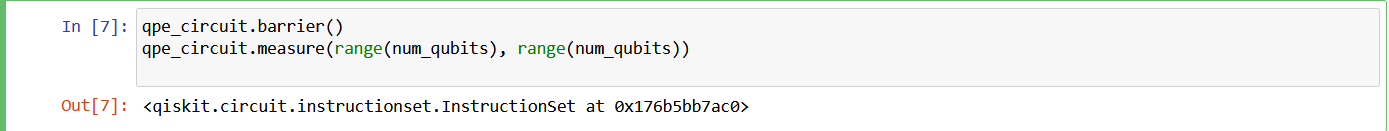
1. Apply controlled unitary operations:



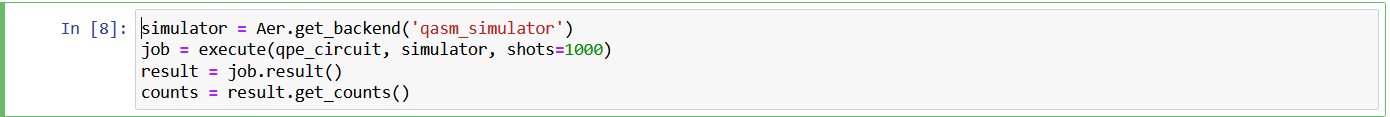
1. Apply inverse quantum Fourier transform (QFT):



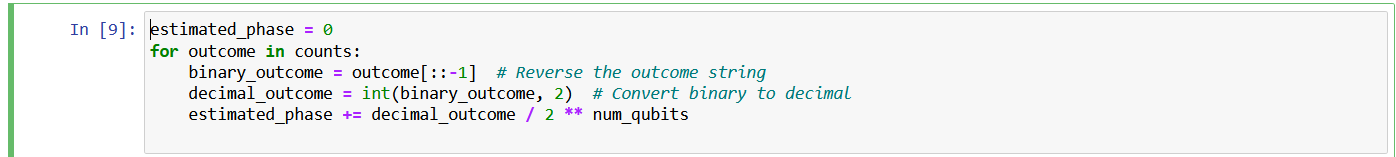
1. Measure the qubits:



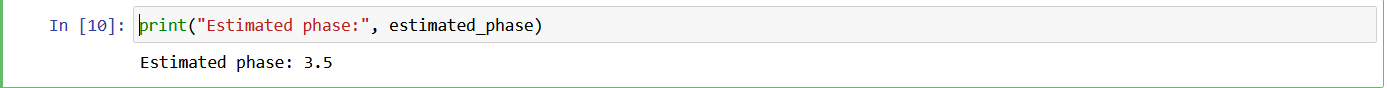
1. Simulate the circuit using Qiskit's Aer simulator:



1. Calculate the estimated phase from the measurement results:



1. Print the estimated phase:



In this example, we define the unitary operator unitary\_matrix with the desired phase to be estimated (theta). We then create a quantum circuit qpe\_circuit for phase estimation with a specified number of qubits and ancillary qubits.

The code applies the necessary gates to perform the Quantum Phase Estimation algorithm, including Hadamard gates, controlled unitary operations, and the inverse Quantum Fourier Transform (QFT). Finally, we measure the qubits and simulate the circuit using Qiskit's Aer simulator. The estimated phase is calculated based on the measurement results.

Finally, we get the estimated phase which is 3.5