Name: _____

Date: _____

Objectives

In this lab, we will delve into the world of quantum computing, exploring its principles, algorithms, and potential applications.

Main Goals:

1. Setup environment
2. Execute Quantum Hello World
3. Analyze Deutsch-Jozsa Algorithm

Setup Environment

In this lab, we will use Google Colab to code Python and run quantum algorithms. We'll install Qiskit directly into the Colab notebook to facilitate this process.

1. Open your Google Drive, click "New," select "More," then "Google Colaboratory" to open a new notebook.
2. Create a new code cell and run the following command to install Qiskit into the Colab environment.

```
!pip install qiskit ipywidgets
```

3. Run the following command to install the Qiskit Aer package which will be used to simulate the quantum circuit.

```
!pip install qiskit-aer
```

4. This is another dependency that needs to get installed to draw the circuit.

```
!pip install pylatexenc
```

5. Run the following lines of code to install the required packages.

```
from qiskit import *
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
%matplotlib inline
```

You have now successfully installed all the required dependencies for the implementation of quantum algorithms.

Implement Quantum Hello World

Run each box of code in *separate* code cells and in the order that they are in:

```
qr = QuantumRegister(2)              # 2-bit quantum register
cr = ClassicalRegister(2)            # 2-bit classical register
circuit = QuantumCircuit(qr, cr)     # build a quantum circuit
circuit.draw('mpl')                  # draw circuit diagram
```

Note: The "mpl" parameter within the draw() function is used to render the circuit using a uniform standard facilitated by Matplotlib. Its primary purpose is to enhance the readability of the circuit diagram.

```
# apply a Hadamard gate to the first qubit
circuit.h(0)
circuit.draw('mpl')
```

What quantum property is associated with the application of a Hadamard gate on a qubit?

```
# apply a CNOT gate to the 2 qubits
circuit.cx(0,1)
circuit.draw('mpl')
```

In this quantum circuit, the control qubit is q0 and the target qubit is q1. What are the possible measurement outcomes of this CNOT gate?

```
circuit.measure(qr, cr)
circuit.draw('mpl')
```

Explain the role of the classical bits in this quantum circuit.

```
# create the simulator
simulator = Aer.get_backend('qasm_simulator')
# prepares the circuit to be executed on the simulator
transpiled = transpile(circuit, simulator)
# executes the transpiled circuit on the simulator
job = simulator.run(transpiled)
# gets results of the execution
result = job.result()
# gets counts of the measurement outcomes
counts = result.get_counts(transpiled)
# plots histogram with counts of the measurement outcomes
plot_histogram(counts)
```

What does the histogram plot represent? Interpret the results of the plot. Does it match what you expected the outcomes to be?

STOP: Before you continue to the rest of the packet, go to File in your Google Colab, download as .py or .ipynb and submit in Moodle.

Understanding the Deutsch-Jozsa Algorithm

*The **Deutsch-Jozsa problem** is figuring out whether a function f is constant (outputting all 0's or all 1's) or balanced (outputting an equal amount of 1's and 0's) by passing an n-bit string and analyzing the outputs.*

You are not required to run this program

```
# Define the Deutsch-Jozsa algorithm function
# Parameters: oracle (constant, balanced) and n (number of bits in str)
# Return quantum circuit
def deutsch_josza_algorithm(oracle, n):
  # Create a quantum circuit with n+1 qubits and n classical bits
  dj_circuit = QuantumCircuit(n+1, n)
  # Apply X gate to the last qubit
  dj_circuit.x(n)
  # Apply Hadamard gate to the last qubit
  dj_circuit.h(n)
  dj_circuit.barrier()
  # Apply Hadamard gates to all the bits
  dj_circuit.h(range(n))
  dj_circuit.barrier()
  # Apply the oracle
  dj_circuit.compose(oracle, range(n+1), inplace=True)
  dj_circuit.barrier()
  # Apply Hadamard  gates to all qubits AGAIN
  dj_circuit.h(range(n))
  dj_circuit.barrier()
  # Measure the first n bits
  dj_circuit.measure(range(n), range(n))
  return dj_circuit

# Define the balanced oracle function
def balanced_oracle(n):
  oracle = QuantumCircuit(n+1)
  for qubit in range(n):
      oracle.cx(qubit, n)
  return oracle

# Define the constant oracle function
def constant_oracle(n):
  oracle = QuantumCircuit(n+1)
  return oracle
```

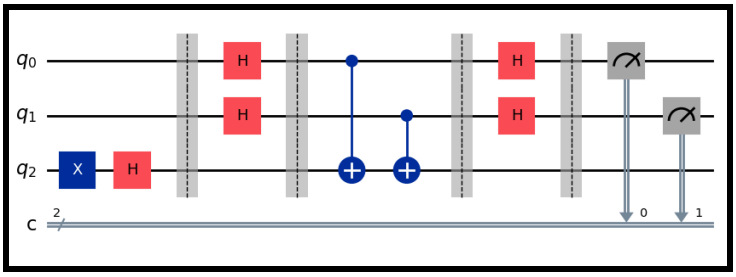
What is the function of an oracle in this algorithm? Explain the difference between the balanced and constant oracle.

You have been given the pre-defined functions that you will need for the Deutsch-Jozsa algorithm. How would you call these functions to answer the Deutsch-Jozsa problem for a 2-qubit string?

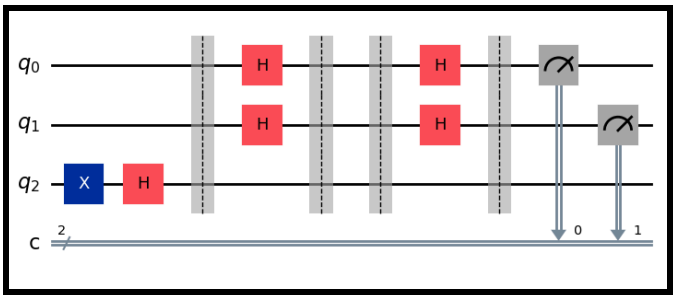
Write the code that will allow you to do the following:

1. Create a variable, `n`, that will equal to 2 to represent the 2-bit input string.
2. Create a variable, `oracle`, that will either be a balanced or constant oracle using `balanced_oracle(n)` or `constant_oracle(n)` respectively.
3. Create a Deutsch-Jozsa circuit using the `deutsch_josza_algorithm(oracle, n)` where the two arguments will be the oracle and n variable you created.
4. Draw the circuit that you created using the `draw()` function.

Depending on which oracle you decide to invoke, these are what your resulting circuits should look like:



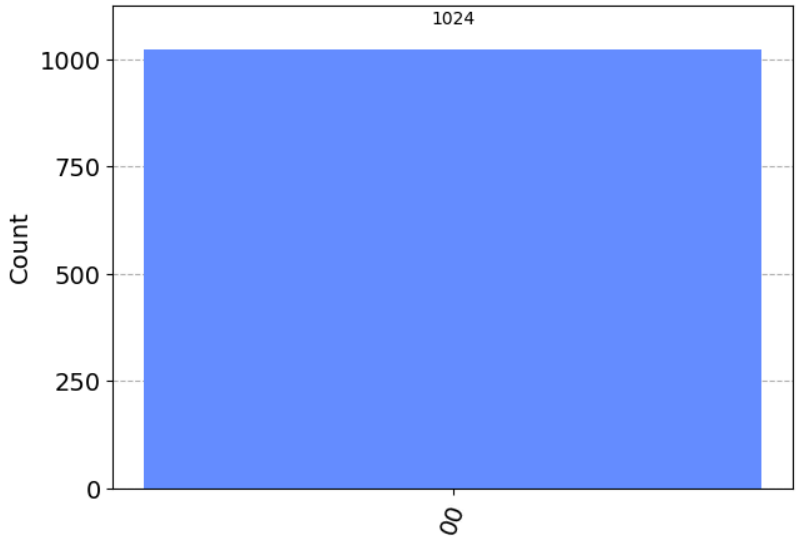
Balanced oracle



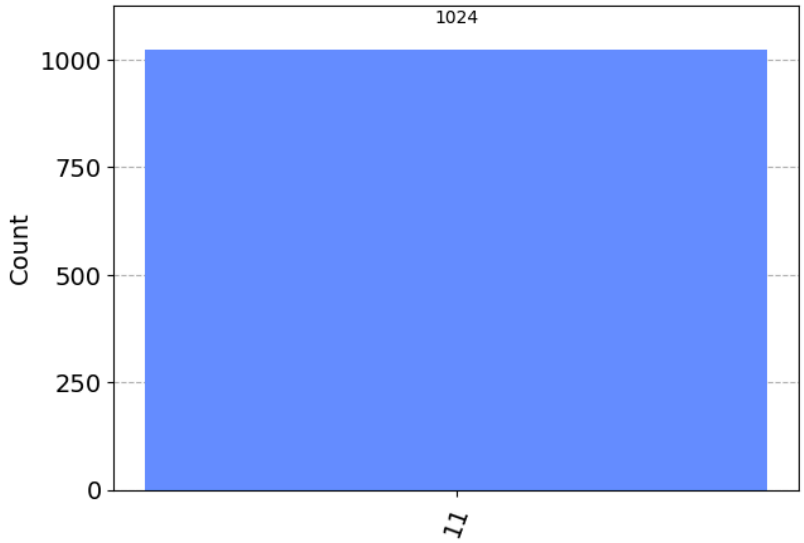
Constant oracle

How does the circuit diagram of a balanced oracle differ from that of a constant oracle in the Deutsch-Jozsa algorithm?

Depending on which oracle you decide to invoke, these are what the measurement outcomes from a quantum simulator should look like:



Constant Oracle



Balanced Oracle

How do the measurement outcomes of a balanced oracle differ from that of a constant oracle in the Deutsch-Jozsa algorithm? Were these the outcomes that we expected?