```
In [ ]: # Initialize Otter
        import otter
        grader = otter.Notebook("q0.ipynb")
```

1 Qiskit Assignment 0

1.1 Introduction to Qiskit

Welcome to your first Qiskit assignment! Qiskit is IBM's open-source SDK for working with quantum computers. We will be using Qiskit for programming assignments this semester. This assignment will help you begin to familarize yourself with the assignment workflow.

1.1.1 Learning Objectives

1. Get familiar with Qiskit
2. Understand the difference between classical and quantum bits
3. Build simple quantum circuits
4. Run your circuit on a quantum computer

1.1.2 Resources

Qiskit assignments are designed to be collaborative and open internet. Where possible, links to helpful resources will be embedded within questions. Generally, you're free to discuss these questions with TAs and peers, but write your own solutions. There may be additional restrictions on, for example, what gates you may be allowed to use to solve each problem. To ensure compliance with course policies and assignment instructions, we reserve the right to inspect your code.

NAME: REPLACE WITH YOUR NAME

STUDENT ID: REPLACE WITH YOUR STUDENT ID

```
In [1]: # Import Qiskit and other needed packages
        from qiskit import *
        from qiskit.visualization import plot_histogram
        from qiskit.quantum_info import Statevector
        from qiskit_textbook.tools import array_to_latex
        import numpy as np
        import pprint
```

Task 1 - Building a Circuit (20 pts) Using Qiskit's QuantumCircuit class, fill in the `simpleCircuit()` function to return a circuit with 1 qubit and 1 classical bit that performs a measurement.

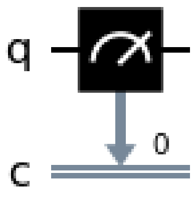
```
In [2]: def simpleCircuit():
            # BEGIN SOLUTION
            qc = QuantumCircuit(1,1)
            qc.measure(0,0)
            return qc
            # END SOLUTION
```

```
In [ ]: grader.check("q1")
```

Task 2 - Drawing Circuits (20 pts) We can visualize circuits using the QuantumCircuit's draw method. Draw your circuit from Task 1 using the matplotlib format.

```
In [7]: # Draw your circuit in this cell

        # BEGIN SOLUTION
        simpleCircuit().draw(output='mpl')
        # END SOLUTION
```

```
Out[7]:
```

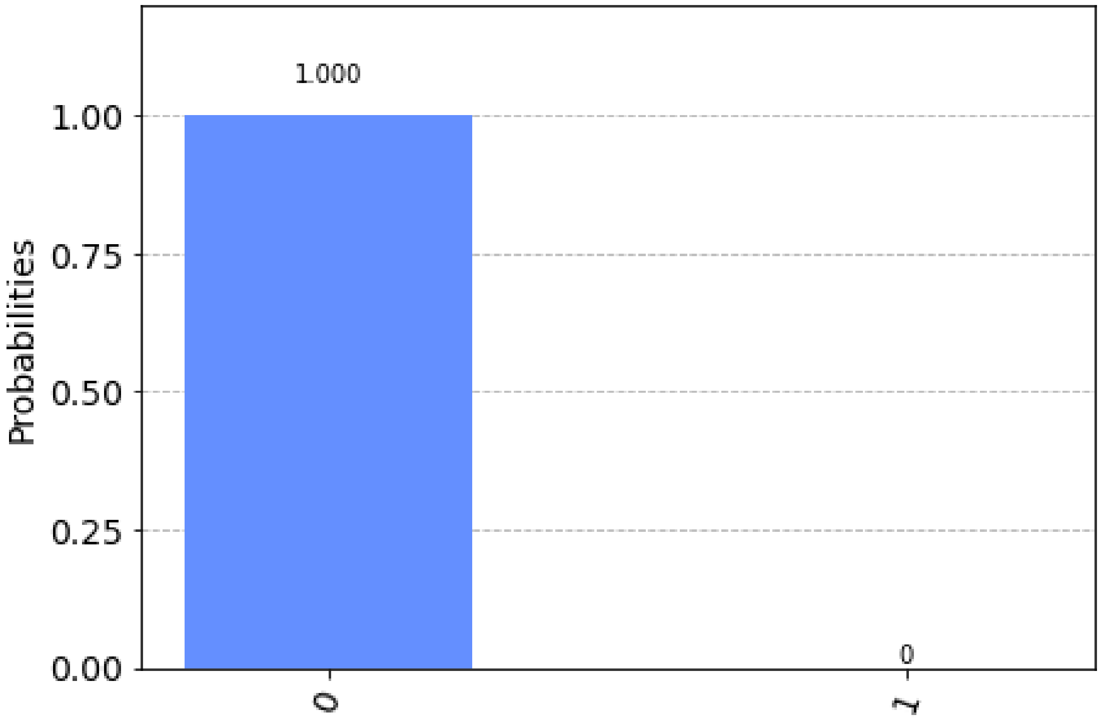


Task 3 - Simulating Circuits and Getting Results (10 pts) Circuits aren't very helpful unless we can run them and observe the outputs. We will use the qasm simulator to simulate our circuit on a quantum machine.

Using the Qiskit docs, execute a job that runs your simpleCircuit 468 times. Do the results match what we'd expect from this circuit?

```
In [8]: qc = simpleCircuit()
        qasm_sim = BasicAer.get_backend("qasm_simulator")

        # BEGIN SOLUTION
        job = execute(qc, qasm_sim, shots=468)
        # END SOLUTION

        counts = job.result().get_counts()

        # This loop displays a state with zero probability
        # on the histogram for the purpose of
        # comparison with the next section.
        for state in ['0','1']:
            if state not in counts:
                counts[state] = 0

        plot_histogram(counts)
```

```
Out[8]:
```



```
In [ ]: grader.check("q3")
```

Task 4 - Running Your Circuit on a Quantum Computer (20 pts) Now let's compare our results from the simulator with the results from a real quantum device. *(What should we expect to see?)*

Create an account with IBM Quantum and paste your API token into the code block below. After running the `save_account` method, you may remove your token to keep it private from Gradescope. Credentials will be saved to your computer and calling `load_account` is sufficient to retrieve them.

```
In [10]: # IBMQ.save_account('replace with your token and uncomment the first time')
```

```
In [11]: IBMQ.load_account()
```

```
Out[11]: <AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

The code block below lists some info about the available IBM quantum devices and queues.

```
In [12]: provider = IBMQ.get_provider(hub='ibm-q')
         for backend in provider.backends():
             status = backend.status().to_dict()
             if status['operational'] and status['status_msg'] == 'active':
                 if 'simulator' not in status['backend_name']:
                     print(pprint.pformat(status))
```

```
{'backend_name': 'ibmq_armonk',
 'backend_version': '2.4.26',
 'operational': True,
 'pending_jobs': 1,
 'status_msg': 'active'}
{'backend_name': 'ibmq_bogota',
 'backend_version': '1.6.16',
 'operational': True,
 'pending_jobs': 38,
 'status_msg': 'active'}
{'backend_name': 'ibmq_lima',
 'backend_version': '1.0.27',
 'operational': True,
 'pending_jobs': 29,
 'status_msg': 'active'}
{'backend_name': 'ibmq_belem',
 'backend_version': '1.0.30',
 'operational': True,
 'pending_jobs': 8,
 'status_msg': 'active'}
{'backend_name': 'ibmq_quito',
 'backend_version': '1.1.20',
 'operational': True,
 'pending_jobs': 72,
 'status_msg': 'active'}
{'backend_name': 'ibmq_manila',
 'backend_version': '1.0.22',
 'operational': True,
 'pending_jobs': 122,
 'status_msg': 'active'}
```

Choose one of the backends from above and insert its name into the code block below. Running this code block will execuete your circuit on an IBM quantum device. **Note: It may take a while for your job to complete based on queue times.** Use the generated link to check your job's status.

```
In [13]: ibmqc = provider.get_backend('replace with a backend_name')
         job = execute(simpleCircuit(), ibmqc, shots=468)
         print("Check job status here:", "https://quantum-computing.ibm.com/jobs/" + job.job_id())
         res = job.result()
         counts = res.get_counts()
         plot_histogram(counts)
```

```
---------------------------------------------------------------------------
QiskitBackendNotFoundError                Traceback (most recent call last)
<ipython-input-13-07ee3753e4aa> in <module>
----> 1 ibmqc = provider.get_backend('replace with a backend_name')
      2 job = execute(simpleCircuit(), ibmqc, shots=468)
      3 print("Check job status here:", "https://quantum-computing.ibm.com/jobs/" + job.job_id())
      4 res = job.result()
      5 counts = res.get_counts()

~/opt/miniconda3/envs/cse468/lib/python3.7/site-packages/qiskit/providers/provider.py in get_backend(s
     53             raise QiskitBackendNotFoundError("More than one backend matches the criteria")
     54         if not backends:
---> 55             raise QiskitBackendNotFoundError("No backend matches the criteria")
     56 
     57         return backends[0]

QiskitBackendNotFoundError: 'No backend matches the criteria'
```

Do you see the same results as the qasm simulator? Why or why not?

Type your answer here, replacing this text.

No. There is error induced by the quantum computer which causes some measurements to yield $|1\rangle$.

Task 5 - Another Circuit (15 pts) We now turn to Qiskit's Pauli X gate so that we can prepare qubits in the $|1\rangle$ state.

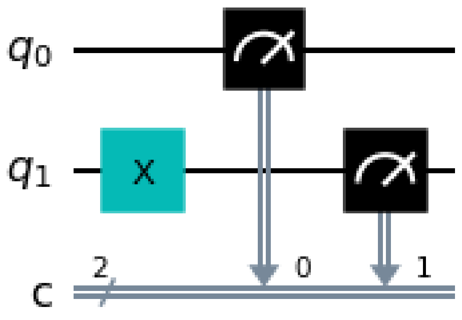
There is also a brief discussion of this gate at the end of `lecture 1: Background`.

- Fill in the below function to return a new QuantumCircuit with 2 qubits and 2 classical bits.
- Prepare the first qubit in state $|0\rangle$ and the second in state $|1\rangle$
- Perform a measurement onto each respective classical bit.
- Draw the circuit using `draw()` and matplotlib

```
In [15]: def opposites():
             # BEGIN SOLUTION
             qc = QuantumCircuit(2,2)
             qc.x(1)
             qc.measure(0,0)
             qc.measure(1,1)
             return qc
             # END SOLUTION
```

```
In [16]: # Draw your circuit in this cell

         # BEGIN SOLUTION
         opposites().draw(output='mpl')
         # END SOLUTION
```

```
Out[16]:
```



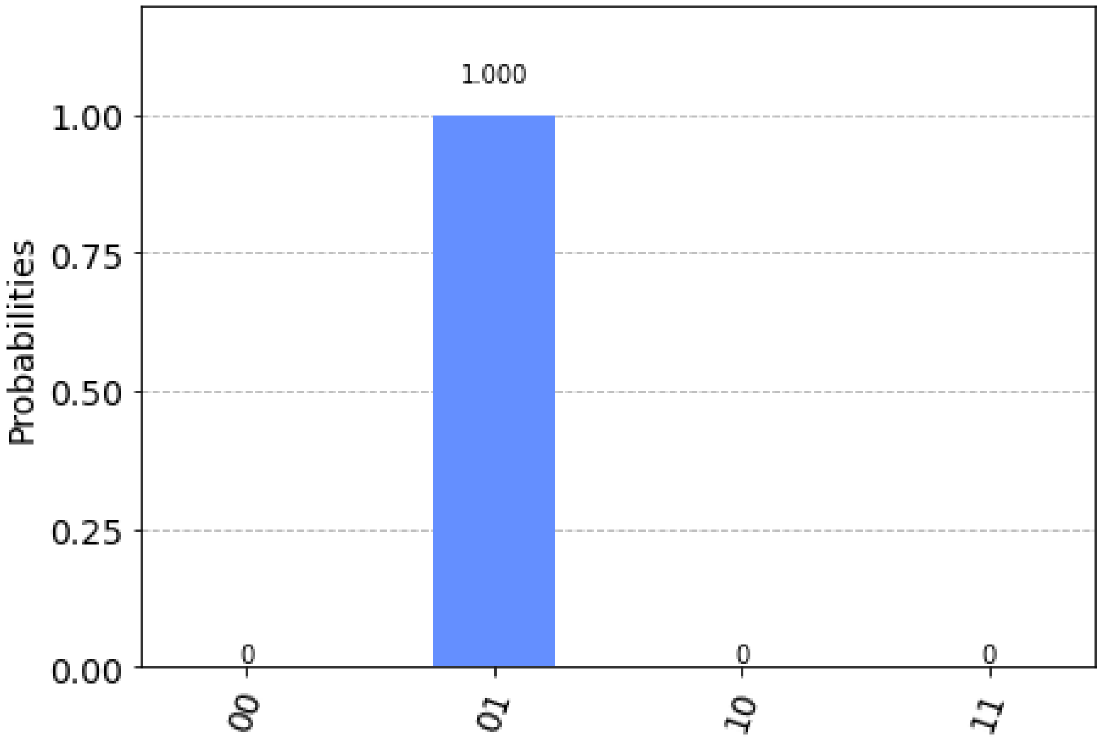
```
In [ ]: grader.check("q5")
```

Task 6 - More Counts (15 pts) Think about what output you expect to see from this circuit. - Run your circuit on the qasm simulator 468 times - Store the measurement results in `counts_oppo`

Note that the following cell uses Qiskit's `reverse_bits()` function. This flips the ordering of qubits a multi-qubit circuit and changes the endianness of the resulting statevector, $|01\rangle$. The reasons for this will become clear in the upcoming lectures and notebook assignments.

```
In [21]: qc = opposites().reverse_bits()
         qasm_sim = BasicAer.get_backend("qasm_simulator")

         # BEGIN SOLUTION
         job = execute(qc, qasm_sim, shots=468)
         counts_oppo = job.result().get_counts()
         # END SOLUTION

         for state in ['00','01','10','11']:
             if state not in counts_oppo:
                 counts_oppo[state] = 0
         plot_histogram(counts_oppo)
```

```
Out[21]:
```



```
In [ ]: grader.check("q6")
```

1.2 Conclusion

This is the general workflow of building and running quantum circuits. We will introduce more qubits and gates as needed to solve problems.

Next time: the unitary gate and single qubit circuits!

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

1.3 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

```
In [ ]: # Save your notebook first, then run this cell to export your submission.
        grader.export(pdf=False)
```