Qiskit Cheat Sheet

We're using Qiskit as a Python library in order to simulate and run quantum circuits. From their website:
"Qiskit [kiss-kit] is an open source software development kit (SDK) for working with quantum computers at the level of pulses, circuits and application modules."

Imports

Import things you'll need for creating and running circuits, and visualizing results:

```
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram
```

Note: In this guide, anything in this blue color is a variable name and can be changed to whatever you want, as long as it's consistent throughout your code

Circuits

Create a circuit with

```
qc = QuantumCircuit(qb, cb)
```

`qc` - the name of the variable the circuit is stored in
`qb` - the number of quantum bits (qubits) in the circuit
`cb` - the number of classical bits; usually the same as the number of qubits, or `0` if using the `statevector_simulator`

Add gates

```
qc.x(qb)
qc.cx(ctrl, target)
```

`qb` - the index of the qubit to apply the X gate to
`ctrl, target` - the indices of the qubits to apply the CNOT gate to

Remember, the qubits are indexed starting from zero

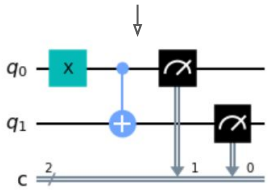
Measure qubits

```
qc.measure(qbits, cbits)
```

`qbits` is a list of the indices of the qubits to measure
`cbits` is a list of the indices of the classical lines which receive the measurements
They should be the same length; see the next panel for an example
This isn't necessary if using `statevector_simulator`

Visualize the circuit

```
qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0,1], [1,0])
qc.draw()
```



In `measure`, the result of measuring the 0^{th} qubit is going to the 1^{st} classical bit (and the 1^{st} qubit to the 0^{th} classical bit)

Recall, this circuit puts the qubits in the $|\phi^+\rangle$ Bell state

Running the circuit

Choose a backend

```
backend = Aer.get_backend(simulator)
```

Common simulators:
- "`qasm_simulator`": ideal and noisy multi-shot execution of circuits; returns counts or memory.
- "`statevector_simulator`": returns the statevector after applying the circuit

Run the job (`shots` only for `qasm_simulator`)

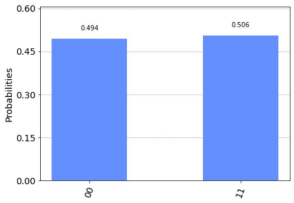
```
job = execute(qc, backend, shots=num_shots)
result = job.result()
```

Get the statevector from `statevector_simulator`

```
state = result.get_statevector(qc)
print(state)  # Display the statevector
```

One way to visualize results

```
counts = result.get_counts(qc)
plot_histogram(counts)
```



This works for both simulators mentioned earlier

Running it on a real quantum computer

```
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=
    lambda x: x.configuration().n_qubits >= 2
    and not x.configuration().simulator
    and x.status().operational==True))
job = execute(qc, backend, shots=num_shots)
job.status() # check the status, it can take a while
```