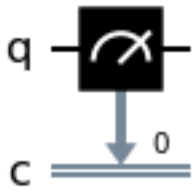
**Task 2 - Drawing Circuits (20 pts)** We can visualize circuits using the QuantumCircuit's draw method.

Draw your circuit from Task 1 using the matplotlib format.

In [9]:
```
# Draw your circuit in this cell

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

Out[9]:



**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.

Create an account with IBM Quantum and paste your API token into the code block below.

In [12]:
```
# IBMQ.save_account('REPLACE WITH YOUR TOKEN AND UNCOMMENT')
```

After running `save_account`, **please delete your token** from the previous cell and save your notebook. This keeps your token private to you and ensures the autograder does not time out. Credentials will be saved to your computer, and calling `load_account` is sufficient to retrieve them for future assignments.

In [13]:
```
IBMQ.load_account()
```

Out[13]:
```
<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 [14]:
```
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_lima',
 'backend_version': '1.0.40',
 'operational': True,
 'pending_jobs': 111,
 'status_msg': 'active'}
{'backend_name': 'ibmq_belem',
 'backend_version': '1.0.53',
 'operational': True,
 'pending_jobs': 6,
 'status_msg': 'active'}
{'backend_name': 'ibmq_quito',
 'backend_version': '1.1.35',
 'operational': True,
 'pending_jobs': 6,
 'status_msg': 'active'}
{'backend_name': 'ibmq_manila',
 'backend_version': '1.1.0',
 'operational': True,
 'pending_jobs': 9,
```

```
 'status_msg': 'active'}
{'backend_name': 'ibm_nairobi',
 'backend_version': '1.2.0',
 'operational': True,
 'pending_jobs': 18,
 'status_msg': 'active'}
{'backend_name': 'ibm_oslo',
 'backend_version': '1.0.14',
 'operational': True,
 'pending_jobs': 20,
 'status_msg': 'active'}
```

We can also use the `least_busy` method to pick the quantum device with the fewest pending jobs.

```
In [15]: lb = least_busy(IBMQ.get_provider(hub='ibm-q').backends(filters=lambda x: not x.configuration()
                                                                 and x.status().operational == True))
         print("Least busy quantum computer:", lb)
```

```
Least busy quantum computer: ibmq_belem
```

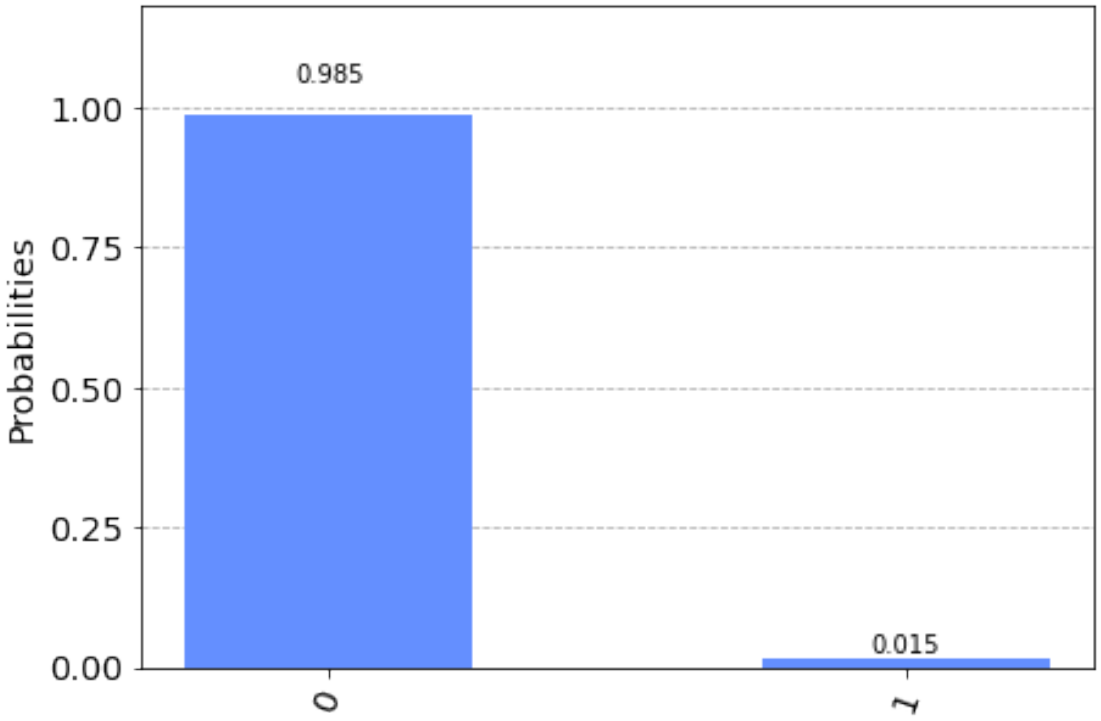
Choose one of the backends from above and insert its name into the code block below. Running this code block will execute your circuit on an IBM quantum device. **Note: It may take a while for your job to complete based on queue times.** Use the generated link to check your job's status.

```
In [17]: 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_ibm = res.get_counts()
         plot_histogram(counts_ibm)
```

```
Check job status here: https://quantum-computing.ibm.com/jobs/639f8dc1a8e4a113e1b3cae2
```

```
Out[17]:
```




Do you see the same results as the qasm simulator? Why or why not? What is the error rate?

*Type your answer here, replacing this text.*

No. There is error induced by the quantum computer's inherent noise which causes some measurements to yield $|1\rangle$. The error rate is about 1.5%.