IBM Q Experience | Quantum Algorit...

Welcome JEAN-MICHEL TORRES

Your accounts

IBM Q Hub France
IBM Q Network member

See more

New here? Get started with the IBM Q Experience!

Circuit Composer

Explore the graphical interface for creating and testing circuits

Create a circuit →

Qiskit Notebooks

Create your first notebook and start using Qiskit

Create a notebook →

Pending results (1)

Status	Id	Account	Backend

Your backends (7)

These are the quantum systems and simulators that you have access to.

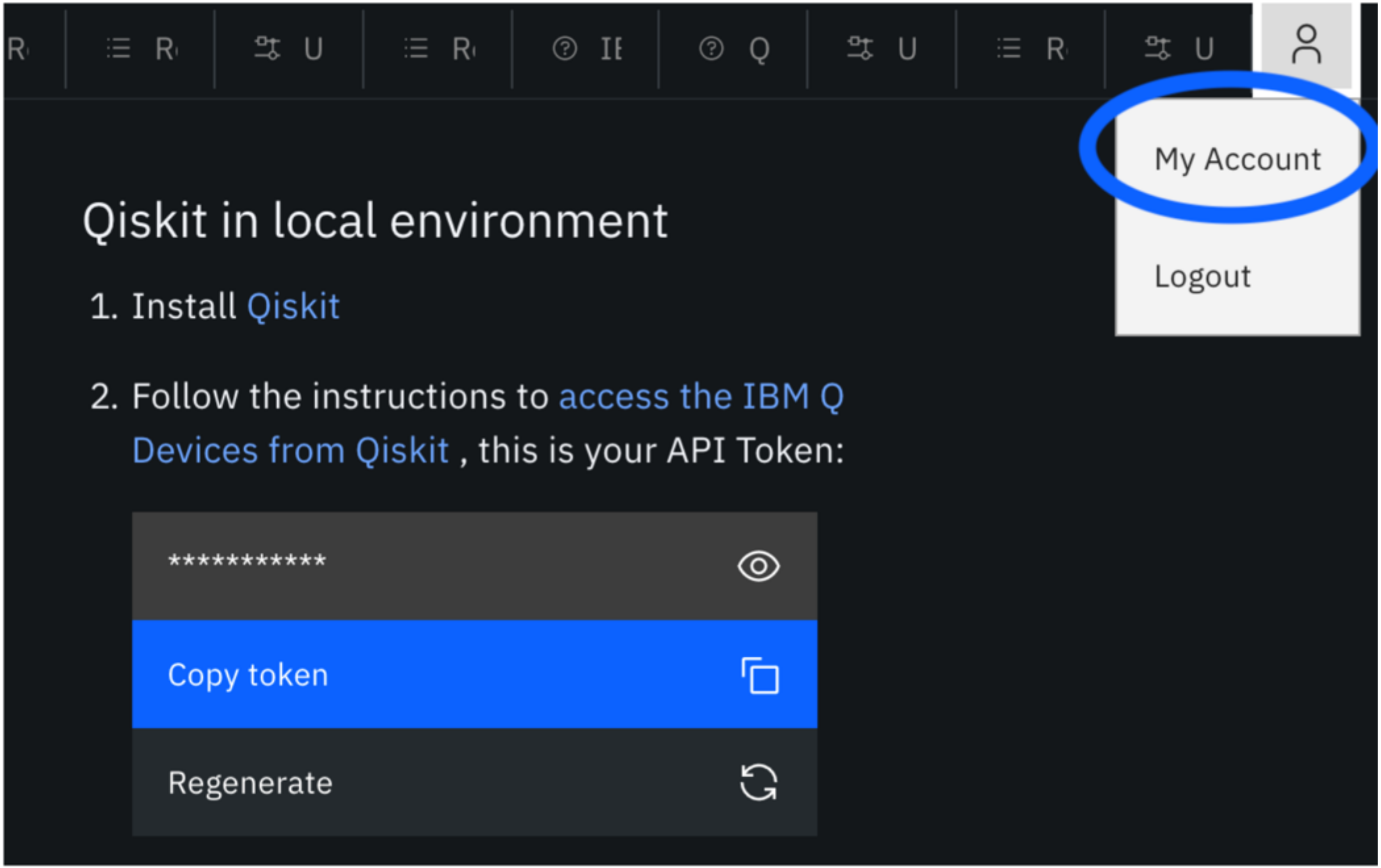
Got it!

online
ibmq_poughkeepsie (20 qubits)
Queue: 0 runs

online
ibmq_20_tokyo (20 qubits)
Queue: 0 runs

online
ibmq_16_melbourne (14 qubits)
Queue: 132 runs

On the upper right corner go to "My Account":



Copy your API Key from here

1. "Hello World" quantum computing with Python and qiskit.

Let's import what we need from qiskit library

- QuantumRegister : define and use qubits register
- ClassicalRegister : to perform measurement into
- QuantumCircuit : to build out circuit
- execute : method for circuit execution
- A backend to execute on, here we are using the local simulator provided within the "Aer" qiskit component
- and tool for results display

In [1]:

```
%matplotlib inline
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit import Aer, execute
backend = Aer.get_backend('qasm_simulator')
from qiskit.tools.visualization import plot_histogram
```

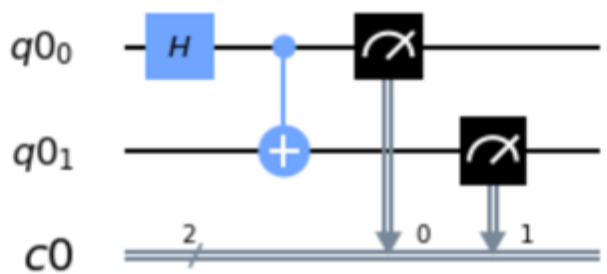
make instance of the required objects, including the quantum circuit, and let's add quantum gates to it:

example : `circ.x(qr[0])` for X gate, `circ.h(qr[0])` for H gate, `circ.cx(qr[0],qr[1])` for CNOT and `circ.measure(qr,cr)` for measurement.

In [2]:

```
# small single qubit circuit
# needed registers
qr = QuantumRegister(2)      # size 2 qubits
cr = ClassicalRegister(2)    # size 2 bits

qc = QuantumCircuit(qr,cr) # circuit using qr and cr

# let's try H and CNOT (Bell state):
qc.h(qr[0])
qc.cx(qr[0],qr[1])

# measurement gate:
qc.measure(qr,cr)

# have a look to check:
qc.draw(output='mpl')
```

Out[2]:

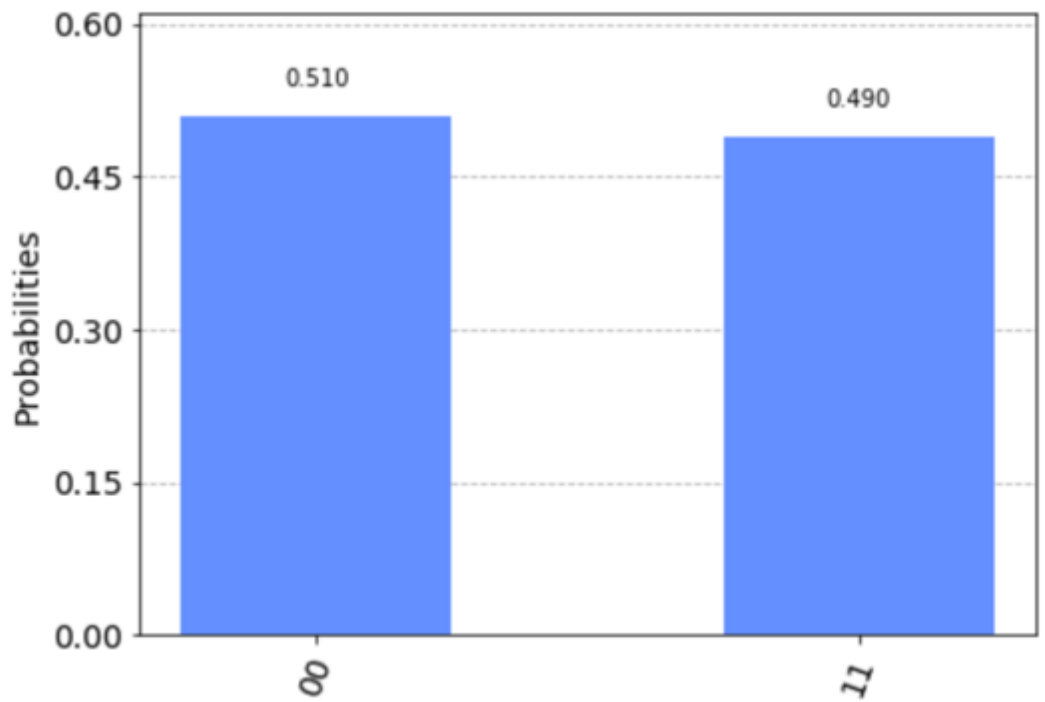


In [3]:
```
# execution and result
resultat = execute(qc,backend,shots=1024).result()

d = resultat.get_counts(qc)
d
```

Out[3]:
```
{'00': 522, '11': 502}
```

In [4]:
```
plot_histogram(resultat.get_counts(qc))
```

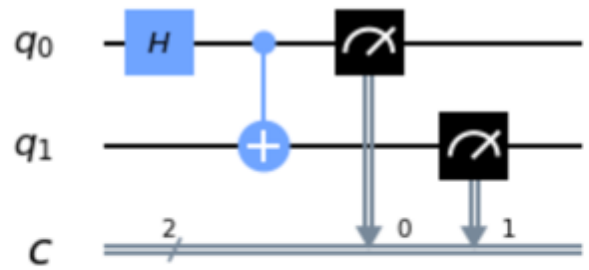
Out[4]:



In [5]:
```
# required librairies
%matplotlib inline
from qiskit import QuantumRegister, QuantumCircuit, ClassicalRegister
from qiskit import execute
```

In [6]:
```
# building Bell state
qc = QuantumCircuit(2,2)

qc.h([0])
qc.cx([0],[1])

qc.measure([0,1],[0,1])
qc.draw(output='mpl')
```

Out[6]:



In [7]:
```
# IBMQ module helps manage your IBM Q account data from your workstation.
from qiskit import IBMQ
```

In [8]:
```
IBMQ.stored_account()
```

Out[8]:
```
{'token': '9[redacted]
3b[redacted],
 'url': 'https://auth.quantum-computing.ibm.com/api'}
```

IBMQ.stored.account() will not work at first time, you need to execute the following (only on time) :

In [ ]:
```
#MY_API_TOKEN= '* * * paste you API token here * * *'
#IBMQ.save_account(MY_API_TOKEN, overwrite=True)
```

In [ ]:
```
# If you had an account activated before qiskit version 0.11 then you need to run this (once for all):
#IBMQ.update_account()
```

In [9]:
```
IBMQ.load_account()
```

Out[9]:
```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

In [10]:
```
# choose one available provider
selected_provider = IBMQ.get_provider(hub='ibm-q')
```

In [11]:
```
# list backends available for this provider
selected_provider.backends()
```

Out[11]:
```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_london') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_burlington') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_essex') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_rome') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

In [12]:

```
# small program to get backends configs and status
# using least_busy() is more straightforward, this is to show
# how we get info from the provider's backends

sp = IBMQ.get_provider(hub='ibm-q')   # selected provider

backends_set = set()
for b in selected_provider.backends():
    backends_set.add(str(b))

print("backend name        queue qubits operational status message")
print("------------------- ----- ------ ----------- --------------")
for b in backends_set:
    be = sp.get_backend(b)
    pj = be.status().pending_jobs
    qb = be.configuration().n_qubits
    op = be.status().operational
    sm = be.status().status_msg
    print(f"{b:20} {pj:4} {qb:6}{op:12} {sm:6}")
```

```
backend name        queue qubits operational status message
------------------- ----- ------ ----------- --------------
ibmq_burlington         4      5           1 active
ibmq_rome               3      5           1 active
ibmq_vigo               3      5           1 active
ibmq_essex              7      5           1 active
ibmq_london             1      5           1 active
ibmq_armonk             2      1           1 active
ibmq_ourense           45      5           1 active
ibmq_qasm_simulator     2     32           1 active
ibmq_16_melbourne       9     15           1 active
ibmqx2                  7      5           1 active
```

In [13]:
```
# choose best backend (can use least_busy() as well ):
backend = sp.get_backend('ibmq_london')
backend.name()
```

Out[13]:
```
'ibmq_london'
```

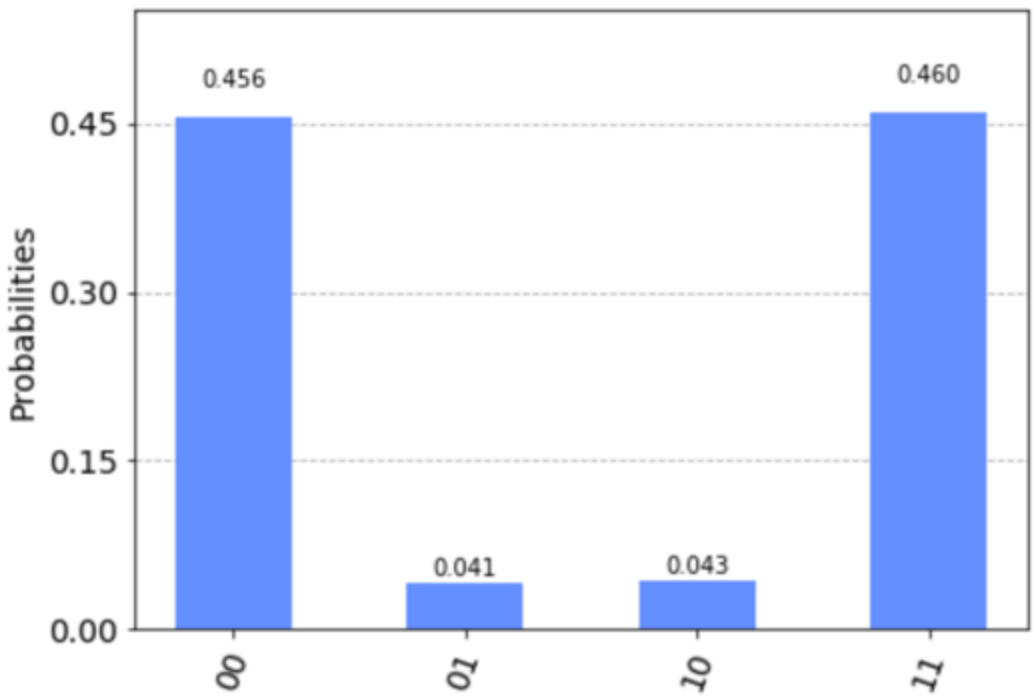
In [14]:
```
# execution

from qiskit.tools.monitor import job_monitor

job = execute(qc,backend, shots=1000)

print(job.job_id())

job_monitor(job)

```

```
5ea9c5f670c4e500180037f2
Job Status: job has successfully run
```

In [15]:
```
## lit le résultat
res = job.result()
```

In [16]:
```
from qiskit.tools.visualization import plot_histogram

d = (res.get_counts(qc))
plot_histogram(d)
```

Out[16]:



In [17]:
```
d
```

Out[17]:
```
{'00': 456, '11': 460, '10': 43, '01': 41}
```