20200108—Qiskit_QCode-1

February 3, 2020

1 20200108—Qiskit_QCode-1

1.0.1 Checking Which Version is Installed

- Since the Qiskit package includes a constellation of different elements, simply printing the version by running qiskit.___version___ can be misleading as it returns only the version for the qiskit-terra package. This is because the qiskit namespace in Python doesn't come from the Qiskit package, but instead is part of the qiskit-terra package.

[1]:
```
import qiskit
```

[2]:
```
qiskit.__version__
```

[2]:
```
'0.10.0'
```

[3]:
```
qiskit.__qiskit_version__
```

[3]:
```
{'qiskit-terra': '0.10.0',
 'qiskit-aer': '0.3.2',
 'qiskit-ignis': '0.2.0',
 'qiskit-ibmq-provider': '0.3.3',
 'qiskit-aqua': '0.6.1',
 'qiskit': '0.13.0'}
```

1.0.2 Tip :

- If you're filing an issue or need to share your installed Qiskit versions for something, use the
- **qiskit__version** attribute

[4]:
```
import numpy as np
from qiskit import(
  QuantumCircuit,
  execute,
  Aer)
from qiskit.visualization import plot_histogram
```

[5]:
```
# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')
# what is Aer ?
```

```
# what is use with Aer.get_backend and to which module this syntax belongs ?
# Why you used 'qasm_simulator' ?
# finally what you are getting and what you understood from this one line␣
 ↪syntax ?
# first of all what do you know with the imported modules and what their␣
 ↪specifications ?

# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)
# What is Quantum Circuit and what the difference with classical circuit ?
# why it has (2,2) and what is first 2 and last 2 and what will happen if␣
 ↪changed the number to 1 or higher than 2 ?
# to which module this QuantumCircuit(x,y) belngs ?

# Add a H gate on qubit 0
circuit.h(0)
# This circuit means QuantumCircuit() so now you took the H gate and what is␣
 ↪h() gate and its importance ?
# how many such gates are there in basic QuantumCircuit() ?
# and did you learnt all the gates which are related to any one algoithm␣
 ↪(schors, grovers etc..) ?


# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)
# what is CNOT gate ?
# in the output you will get X image and are these both same or not. if not␣
 ↪then what is the difference ?
# cx(0,1) here 0 is control qubit and why its named as control and what it can␣
 ↪do the control ?
# cx(0,1) here 1 is target qubit and in this syntax only we should put the␣
 ↪target qubit or any other syntax do we have ?
# finally what we are getting from this syntax ?

# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])
# so this syntax is going to measurement and now what does it mean i mean what␣
 ↪is quantum measurement ?
# what do you calculating here ?
# why [0,1] for 2 times in this syntax ?
# finally what we are getting from this syntax ?

# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1000)
# here look at the first qasm_simulator and this belongs to Aers, here you␣
 ↪going to executing in this simulator
```

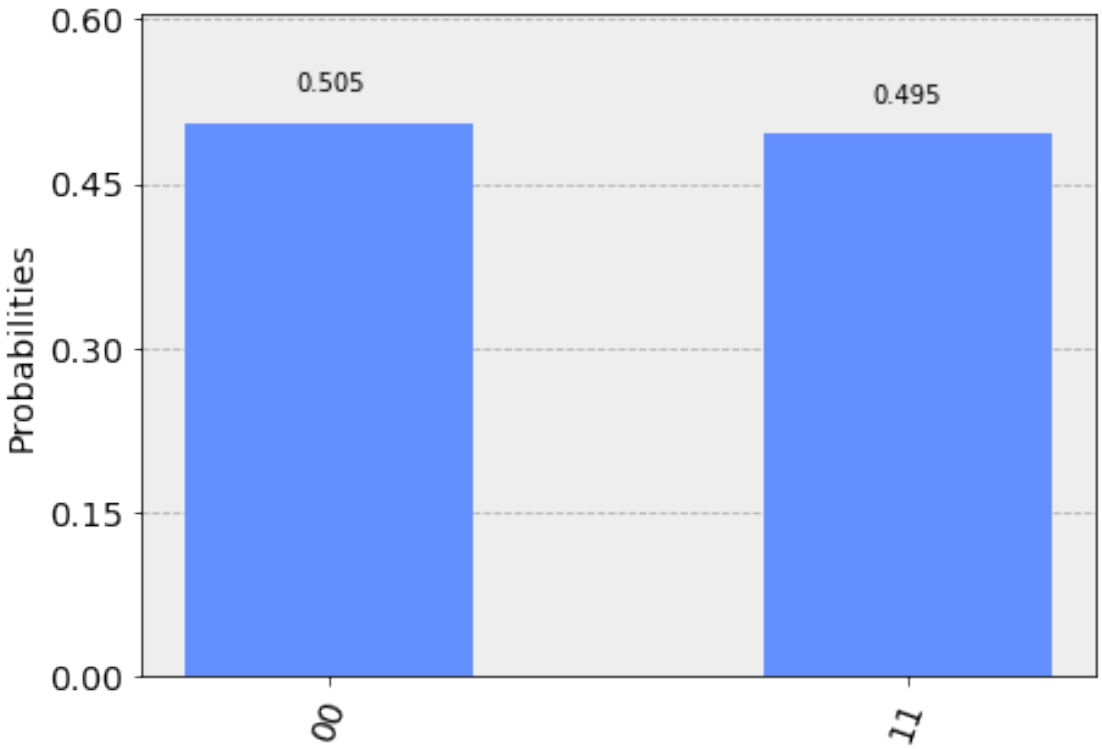
```
# How many other simulators are there in Aers and what their uses and whats use␣
 ↪did you find with this simulator ?
# what does it mean shots and why 1000 and can you use more than 1000 or lower␣
 ↪or whats use with this and losses if not ?
# finally what we are getting from this syntax ?

# Grab results from the job
result = job.result()

# Returns counts
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
# did you see that the output of counts is in DICTIONARY format and why and 00,␣
 ↪11 and why not 01 and 10 ?
# finally what we are getting from this syntax ?

# Draw the circuit
circuit.draw()
```

```
Total count for 00 and 11 are: {'00': 505, '11': 495}
```

[5]:
```
<qiskit.visualization.text.TextDrawing at 0xd9ef1c8>
```

[6]:
```
# Plot a histogram
import matplotlib.pyplot as plt
plot_histogram(counts)
```

[6]:



2 Work Flow Step-by-Step

- to understand the above program then do the steps as
  - Import packages (no need if already imported)
  - Initialize the variables
  - Add gates
  - Visualize the circuit
  - Simulate the experiment
  - Visualize the result

[7]:
```
# Step:2 - Initialize the variabels
circt = QuantumCircuit(2, 2)
# syntax = QuantumCircuit(int,int)
# Here, you are initializing with 2 qubits in the zero state; with 2 classical␣
 ↪bits set to zero; and
# circuit is the quantum circuit.
```

[8]:
```
# Step:3 - Add gates
# gates (operations) are to manipulates the registers of your circuit
# consider the three lines code
circt.h(0)
circt.cx(0,1)
```

```
circt.measure([0,1],[0,1])
```

[8]: <qiskit.circuit.instructionset.InstructionSet at 0x13cf22c8>

- In the above three line code - The gates are added to the circuit one-by-one to form the Bell state
- |psi> = (|00>+|11>)/sqrt(2)
- The code above applies the following gates:
- QuantumCircuit.h(0) : A Hadamard gate (H) on qubit 0, which puts it into a ``superposition state.''
- QuantumCircuit.cx(0, 1) : A controlled-Not operation (Cx) on control qubit 0 and target qubit 1, putting the qubits in an ``entangled state.''
- QuantumCircuit.measure([0,1], [0,1]) : if you pass the entire quantum and classical registers to measure , the ith qubit's measurement result will be stored in the ith classical bit.

[9]:
```
# Step4: Visualize the circuit
# You can use QuantumCircuit.draw() to view the circuit that you have designed␣
 ↪in the
# various forms used in many textbooks and research articles.
circt.draw()
```

[9]: <qiskit.visualization.text.TextDrawing at 0x13cf2888>

- In this circuit, the qubits are ordered with qubit zero at the top and qubit one at the bottom. The circuit is read left-to-right, meaning that gates which are applied earlier in the circuit show up farther to the left.
- The default backend for QuantumCircuit.draw() or ``qiskit.visualization.circuit_drawer()'' is the text backend. However, depending on your local environment you may want to change these defaults to something better suited for your use case. This is done with the user config file.
- By default the config file should be located in ~/.qiskit/settings.conf and is a .ini file
  - for example a settings.conf file for setting a Matplotlib drawer is
  - [default]
  - circuit_drawer = mpl
- You can use any of the valid circuit drawer backends as the value for this config, this includes text, mpl, latex, and latex_source.

[10]:
```
# STep:5 - Simulate the experiment
# Qiskit Aer is a high performance simulator framework for quantum circuits.
# It provides several backends to achieve different simulation goals.
# If you have issues installing Aer, you can alternatively use the Basic Aer␣
 ↪provider
# by replacing Aer with BasicAer. Basic Aer is included in Qiskit Terra.
# import numpy as np
```

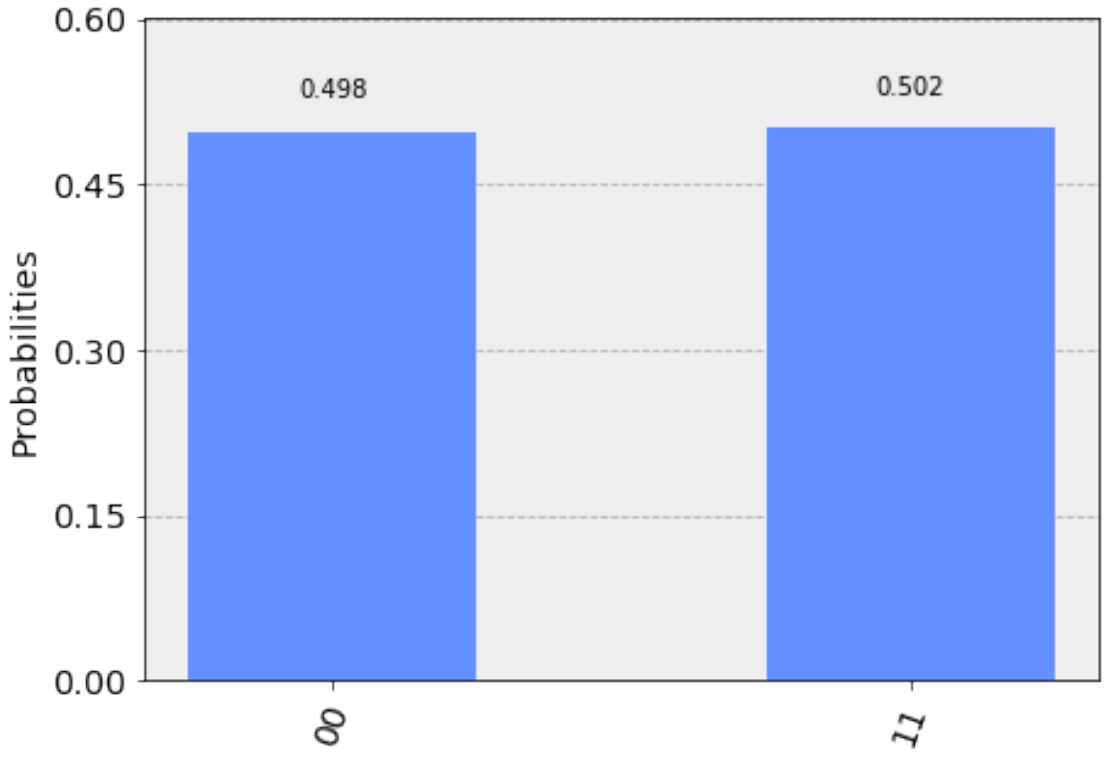
```
# from qiskit import(QuantumCircuit,execute,BasicAer)
# To simulate this circuit, you will use the qasm_simulator . Each run of this
 ↪circuit
# will yield either the bit string 00 or 11.
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1000)
result = job.result()
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
```

```
Total count for 00 and 11 are: {'00': 498, '11': 502}
```

- As expected, the output bit string is 00 approximately 50 percent of the time. The number of times the circuit is run can be specified via the `shots' argument o the execute method. The number of shots of the simulation was set to be 1000 (the default is 1024).
- Once you have a `result' object, you can access the counts via the method `get_counts(circuit)' . This gives you the aggregate outcomes of the experiment you ran.

[11]:
```
# Step:6 - visualize the results
# Qiskit provides many visualizations, including the function "plot_histogram"
 ↪, to view your results.
plot_histogram(counts)
```

[11]:



- The observed probabilities Pr(00) and Pr(11) are computed by taking the respective counts and dividing by the total number of shots.
- Note
  - Try changing the ``shots'' keyword in the ``execute'' method to see how the estimated probabilities change.

[12]:
```
# ------------------------------- END ------------------------------
```

3 Next Steps

Now that you have learnt the basics, consider these learning resources:

Notebook tutorials Video tutorials API References

[ ]:
```

```