Quick Guide

Institute of Quantum Computing, Baidu.

September 9, 2020

1 Overview

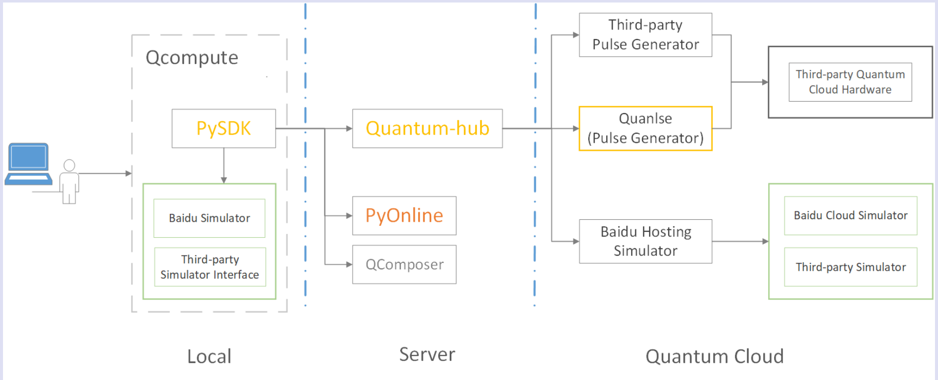
Introduction

Quantum Leaf is a cloud-native quantum computing platform developed by a relevant research subsidiary of Baidu Research Institute, the Quantum Computing Institute, that provides a quantum computing environment based on the QaaS model (Quantum infrastructure as a service) . Users can program on Quantum Leaf, and choose to run quantum programs on a quantum simulator or a quantum computer. In order to better help users perform quantum computing experiments and algorithm research, Quantum Leaf is divided into the following three parts based on the diversified demands of users:

1. Quantum online platform: PyOnline
   PyOnline is easy to install and can be used for non-engineering research and exploration. Users can enjoy smooth and rich quantum experience by merely logging-in to Quantum-hub. PyOnline adopts a containerized attach operation mode and supports self-defined files as well as Python components, enabling users to make use of the power of quantum computing accurately and flexibly.
2. Quantum composer: QComposer
   QComposer, designed for beginners and advanced users, allows users to construct quantum circuits and perform verification by simply dragging the corresponding components, leaving out the complexity of coding. The web end will display the corresponding codes based on the completed quantum circuits to help users learn quantum programming.
3. Quantum development kit: QCompute
   QCompute can provide full-stack end-to-end quantum services for users (mainly computer engineers and scientists) , allowing them to use the Python programming language to run quantum programs on a simulator hosted on the user's device, or on a remote quantum computer. The simulator hosted on the user' s device could be a Baidu simulator or a third-party simulator. Users can use it locally or remotely by establishing a connection with an online simulator. To connect to the online simulator, users should have a Quantum-hub account and use the account-corresponding Token.

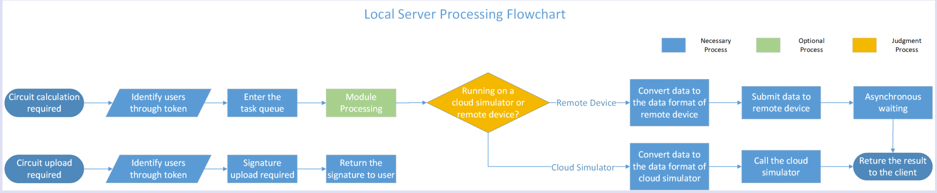
“ *Quantum mechanics: Real Black Magic Calculus* (Albert Einstein) ”

Architecture of Quantum Leaf

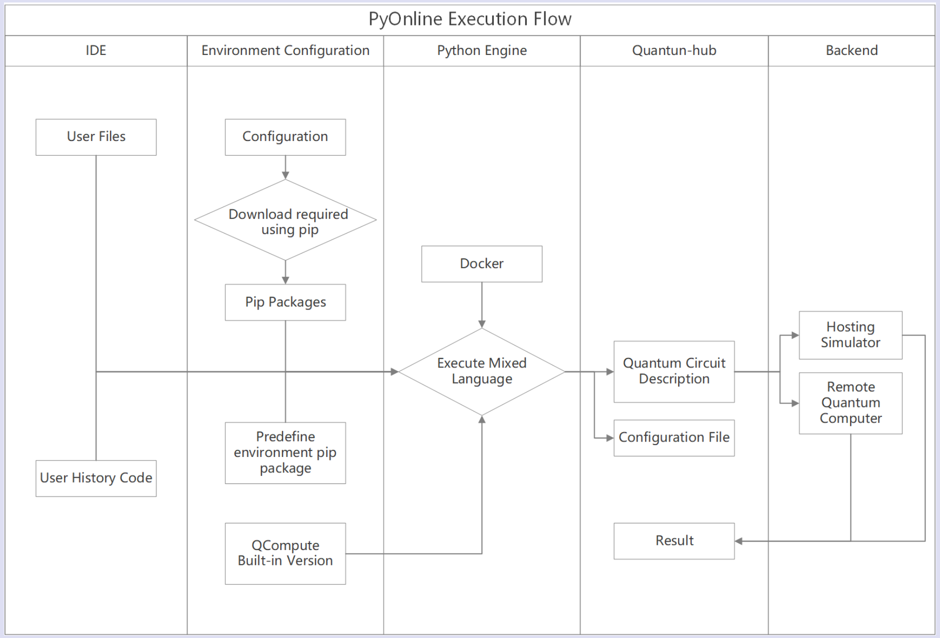


1. Three types of components are available for the front end in programming.
   (a) The convenient PyOnline: PyOnline is an online Python editor equipped with SDK components that can be immediately used by simply logging in, saving you from the troubles of configuration. Meanwhile, with its highly efficient extendable architecture, PyOnline is committed to protecting users' data and knowledge assets, ensuring that users can upload self-defined files, download generated file, and define the installation packages that need to be used.
   (b) The simple QComposer: QComposer allows you to effectively construct quantum circuits by dragging-and-dropping quantum gate icons or performing QASM interactive programming.
   (c) The flexible QCompute: QCompute boasts the most comprehensive quantum computing capability, as it carries not only a lightweight quantum simulator (Baidu Sim2) developed by Baidu independently, but also a third-party simulator and an interface for connecting remote quantum cloud hardware. With the help of Python's flexible syntax and various other functions, users' knowledge assets can be managed and reused, and users can save codes (circuits) locally or upload them to the cloud.
2. The quantum end provides a variety of quantum service forms
   (a) **LocalBaiduSim2** State Vector simulator Sim2 (developed by Quantum Computing Institute Baidu with Python programming language) runs locally.
   (b) **CloudBaiduSim2** State Vector simulator Sim2 runs in a cloud container (Quantum-hub) and is supported of the computing power of Baidu Cloud.
   (c) **CloudIbmAerAtBD** The old open-source IBM Aer simulator (CPP version) runs in a cloud container (Quantum-hub), and is support of the computing power of Baidu Cloud.
   (d) **VIP has more benefits, such as using the QPU for testing purpose.**

Client workflow



PyOnline Workflow

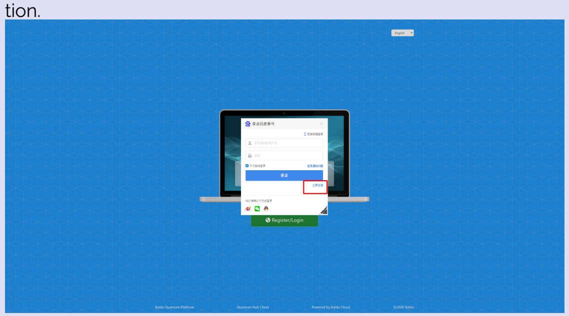


2 Quick Start

User Account and Token

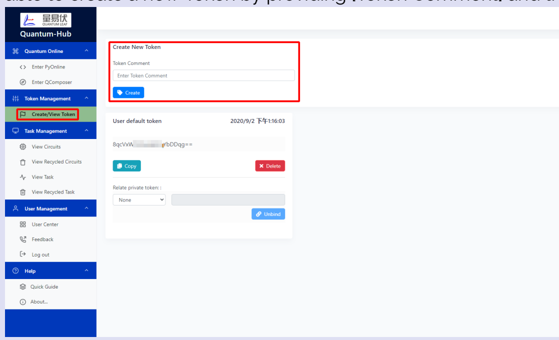
Three types of components are available for quantum computing programming. Users can use QCompute locally or remotely by establishing a connection with an online simulator, while PyOnline and QComposer need to logging-in to Quantum-hub. In the following introductory content, users will be guided to log in to Quantum-hub and generate Token (the default Token is generated after logging in), then get start with PyOnline, QComposer and QCompute.

1. Signup and Login
   Visit Quantum-hub in browser and click [Register/Log in] on the bottom of the page to open the log-in window. First-time users please click [Signup Now] and follow all the steps to complete the registration.



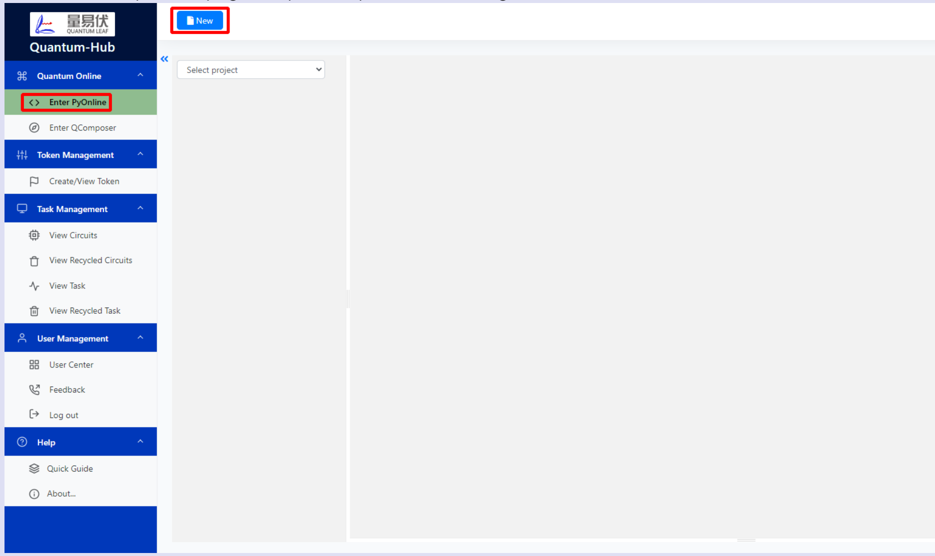
   After signing up, please return to Quantum-hub's homepage and log in.

2. Generate a Token
   The first step to use Quantum-hub is to generate a Token. A Token is like an ID, which is used by Quantum-hub to verify the identity of a user. When users want to submit their local codes to Quantum-hub's cloud server for execution, they need to include their Tokens in the codes which can be obtained from Quantum-hub.
   After logging in, click [Create/View Token] on the sidebar to switch to the the Token-management interface. Every user has a default Token, consisting of letters, numbers, and equal signs. Users are also able to create a new Token by providing [Token Comment] and then click [Create].

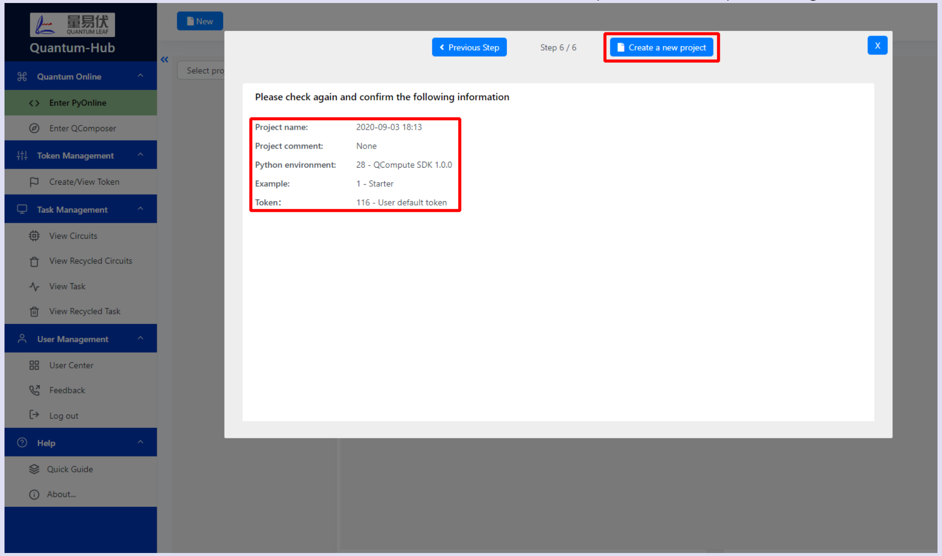


Getting Started with PyOnline 01

After logging in, click [Enter Quantum Online] on the sidebar to switch to the online project interface. Click [New] on the top of the page to open the project creation guide.

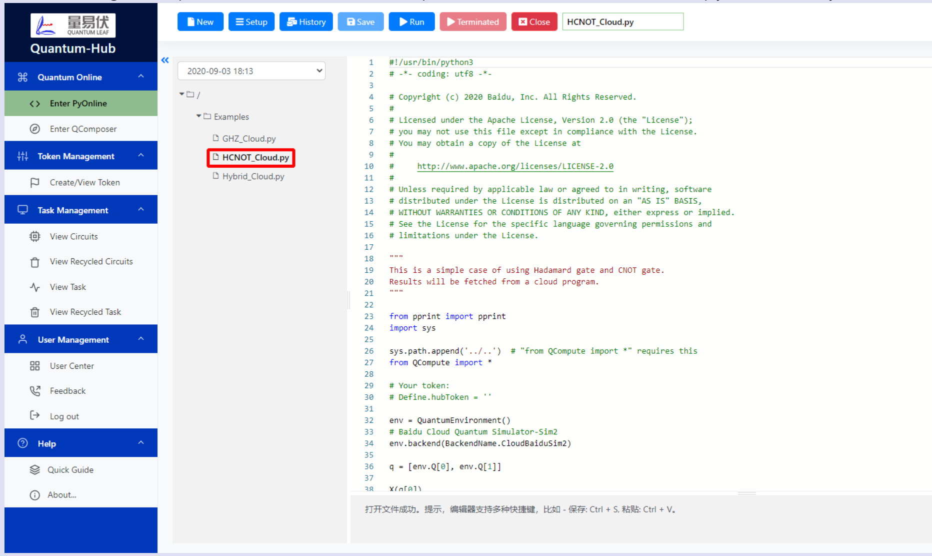


This guide consists of six steps. Users need to follow the prompts and complete all the steps. In step 6, please check all the information filled in before and then click [Create a new project] to complete the guide.



Getting Started with PyOnline 02

After creating a new project, there are three examples. Double click HCNOT-Cloud.py to view its Python codes.

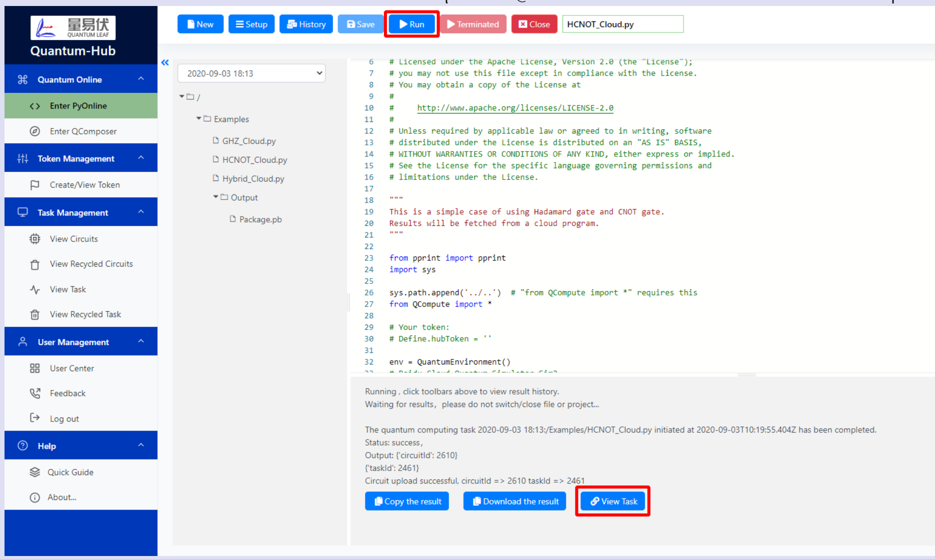


The comments below explain how the codes work.

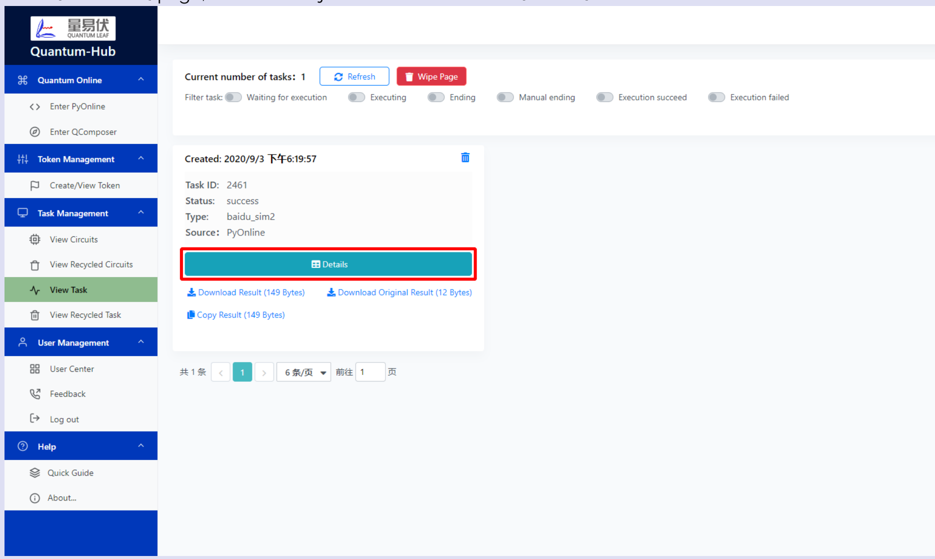
```python
from QCompute import *

# Create a new quantum environment
env = QuantumEnvironment()
# Use the quantum simulator developed by Baidu
env.backend(BackendName.CloudBaiduSim2)

# Initialize two qubits at state |00>
q = [env.Q[0], env.Q[1]]
# Add an X gate on qubit 0 (the first qubit)
X(q[0])
# Add a CNOT gate with qubit 0 and qubit 1 being the control qubit and the target qubit, respectively
CX(q[0], q[1])

# Measure both qubits
MeasureZ(q, range(2))
# Measure the circuit for 1024 times. The result does not return, please view it in [View Circuits] and [View Task]
taskResult = env.commit(1024, downloadResult=False)
```

Getting Started with PyOnline 03

Click [Run] on the top of the page. After a while, there will appear three buttons below the codes. Click [View Task] to jump to the page for viewing tasks. Note that everytime a user uses online computational resources, it will cost 1 Credit in their account. Please contact quantum@baidu.com when all the Credits are used up.

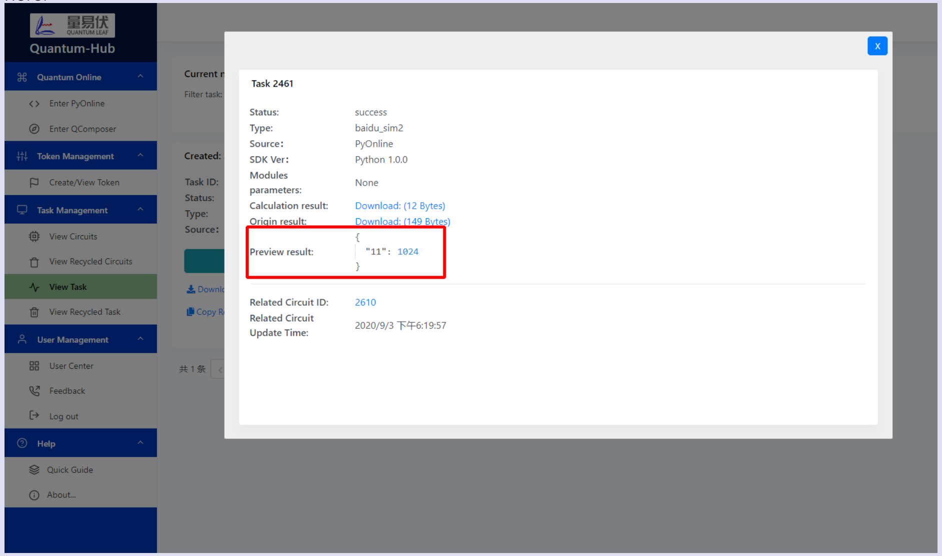


In the [View Task] page, find the task just executed and click [Details] for measurement results.

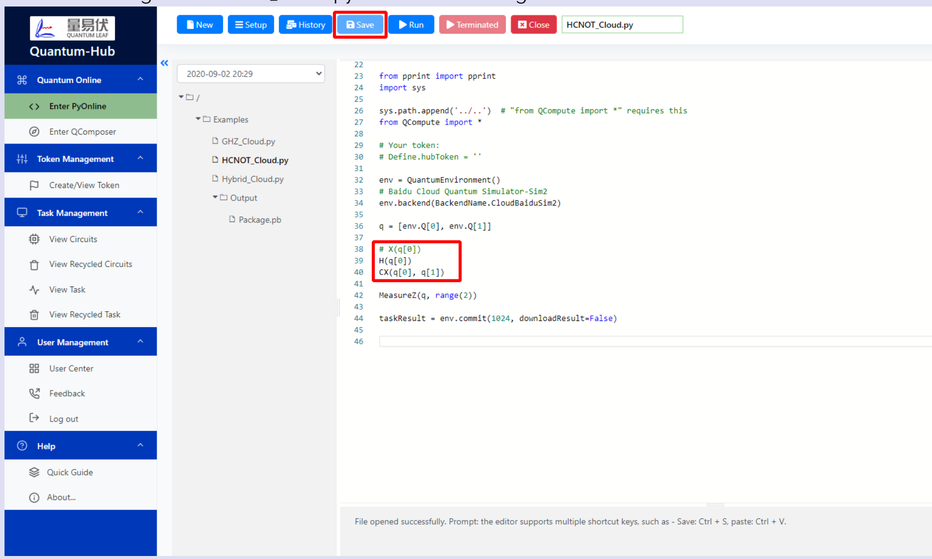


Getting Started with PyOnline 04

As shown in the figure, all measurements give the same result, 11, which corresponds to state $|11\rangle$. Note that currently Baidu's quantum circuit simulation includes no noise, and thus ideal results here can be obtained here.

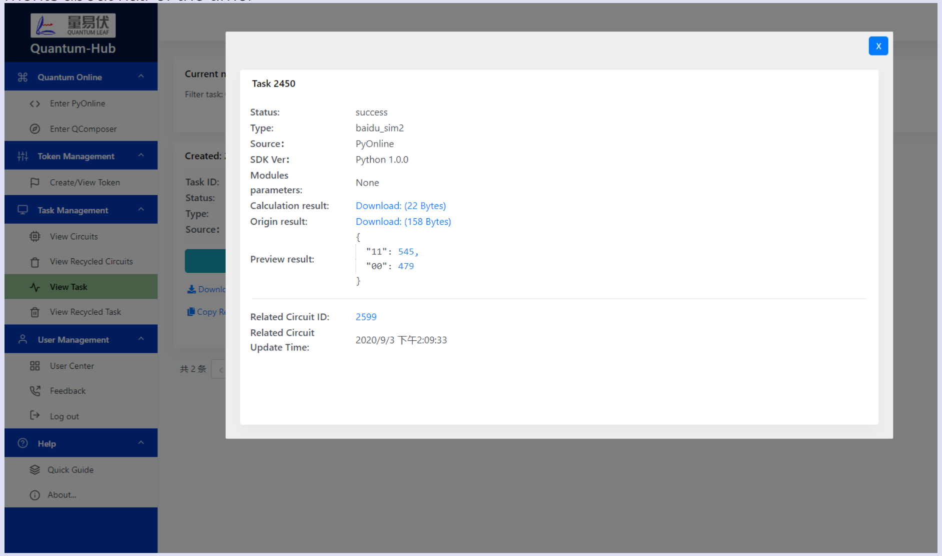


Now, let's try to run the circuit for Bell state preparation. Back to the online project interface, replace the X gate with Hadamard gate in HCNOT_Cloud.py and save the change.



Getting Started with PyOnline 05

By runing the codes and checking the results, there are both states $|00\rangle$ and $|11\rangle$ are obtained by measurements about half of the time.

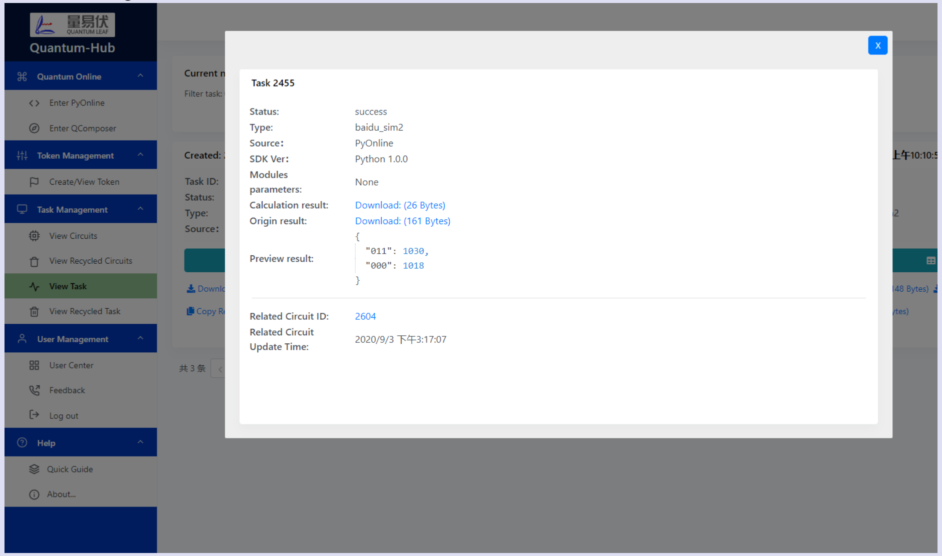


The next example shows how to use the measurement function 'MeasureZ()'. Users can use this function to measure selected qubits. For example, the codes below initialize 6 qubits, but only take three qubits, qubits 2, 4, and 5, for measurement.

```python
from QCompute import *

# Create a new quantum environment
env = QuantumEnvironment()
# Use the quantum simulator developed by Baidu
env.backend(BackendName.CloudBaiduSim2)

# Initialize six qubits at state |000000>
q = [env.Q[i] for i in range(6)]
# Add a Hadamard gate on qubit 2 (the third qubit)
H(q[2])
# Add a CNOT gate with qubit 2 and qubit 4 being the control qubit and the target qubit, respectively
CX(q[2], q[4])

# Measure qubits 2, 4, and 5; the number 3 in range(3) represents the number of qubits to be measured
MeasureZ([q[2], q[4], q[5]], range(3))
# Measure the circuit for 2048 times. The result does not return, please view it in [View Circuits] and [View Task]
taskResult = env.commit(2048, downloadResult=False)
```

Getting Started with PyOnline 06

In theory, about half of the measured quantum state are $|000\rangle$ and the other half are $|110\rangle$, while the results from executing the codes show that both 000 and 011 are returned for about half of the time.



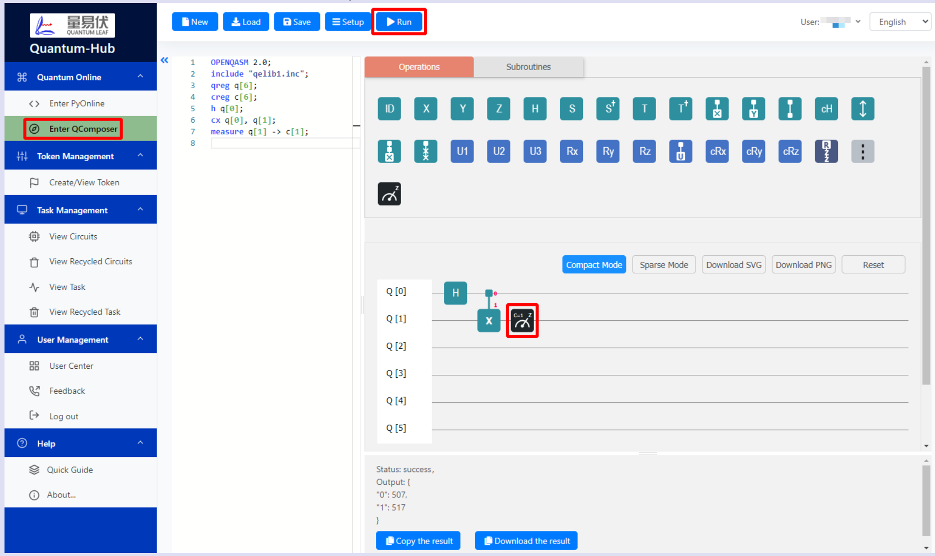
In fact, 011 in [Preview result] is equivalent to state $|110\rangle$. Most textbooks use big-endian, i.e., $|q_0q_1q_2\cdots\rangle$, while Quantum-hub uses little-endian when outputing results, i.e., $q_nq_{n-1}q_{n-2}\cdots$. Users can try the example below for a better understanding.

```python
from QCompute import *

# Create a new quantum environment
env = QuantumEnvironment()
# Use the quantum simulator developed by Baidu
env.backend(BackendName.CloudBaiduSim2)

# Initialize six qubits at state |000000>
q = [env.Q[i] for i in range(6)]
# Add an X gate on qubit 0 (the first qubit)
X(q[0])

# Measure all the qubits
MeasureZ(q, range(6))
# Measure the circuit for 2048 times. The result does not return, please view it in [View Circuits] and [View Task]
taskResult = env.commit(2048, downloadResult=False)
```

Running the codes above, users will get 2048 measurement results all being 000001, corresponding to state $|100000\rangle$.

Getting Started with QComposer

Click [Enter QComposer] on the sidebar to switch to the QComposer interface, where users can add quantum gates to a quantum circuit by dragging icons. After constructing a circuit, click [Run] to measure the circuit. **Note** that a measurement block is required for succesful execution.



Above is a quick guide to Quantum-hub. Users can also try some more interesting examples, e.g., VQE (Variational Quantum Eigensolver) and Grover's algorithm.

Getting Started with QCompute 01

QCompute is an open-source quantum computation tool that can be used locally, empowered by Baidu cloud computational resources.

1. Install Python Environment
   Install Python 3.6 or later versions.
2. Use PIP to Install QCompute
   Run

   ```
   pip install qcompute
   ```

   or

   ```
   pip install -e .
   ```

3. Verify Installation
   After installing QCompute, run the code below to ensure that it is correctly installed.

   ```
   python -m QCompute.Test.PostInstall.PostInstall
   ```

   After a while, if it outputs:

   ```
   Local test succeeded.
   ```

   the local simulators are ready to use. When it prompts:

   ```
   Please provide a token:
   ```

   please enter users' Token obtained from Quantum-hub . After a while, if it outputs:

   ```
   Cloud test succeeded.
   ```

   the cloud simulators are ready to use.

Getting Started with QCompute 02

After creating a new Python file called bell, import packages in need.

```
from pprint import pprint
import sys
sys.path.append('../..')  # from QCompute import *; Needing
from QCompute import \*
```

Define Token, **Token uniquely identifies the circuit submitter. Users should include it in code when use QCompute to connect to remote simulators or devices for experiments. PyOnline does not require users to fill in by themselves. QPU is for VIP users only.**

```
Define.hubToken = ''
```

Set an environment.

```
env = QuantumEnvironment()
# Baidu Cloud Quantum Simulator-Sim2
env.backend(BackendName.CloudBaiduSim2)
```

Set qubits.

```
q = [env.Q[0], env.Q[1]]
```

Construct a quantum circuit.

```
H(q[0])
CX(q[0], q[1])
MeasureZ(q, range(2))
```

Submit the circuit. Fetch the result and print it.

```
taskResult = env.commit(1024, fetchMeasure=True)
pprint(taskResult)
```

The preparation of the Bell state is finished now.

Quantum-hub or QCompute

Quantum-hub is a central console of Quantum Leaf. The results can be obtained from Quantum-hub no matter whether the PyOnline, QComposer running in the cloud or the QCompute running locally. QCompute through which users can program in Python and simple quantum programming language, is a high-level development component of Quantum Leaf. PyOnline deployed in the cloud also contains components of QCompute. An example of a Bell state circuit measurement is shown below.

First, we use Baidu cloud server to run the following code. Each execution cost user 1 Credit. Note that when using cloud resources, a user must first login to Quantum-hub, copy their Token, and paste it into the line Define.hubToken = ''.

```
from pprint import pprint
import sys
sys.path.append('../..')
from QCompute import *

Define.hubToken = 'Put Token here'
# Create a new quantum environment
env = QuantumEnvironment()
# Use the cloud quantum simulator developed by Baidu
env.backend(BackendName.CloudBaiduSim2)

# Initialize two qubits at state |00>
q = [env.Q[0], env.Q[1]]
# Add a Hadamard gate on qubit 0 (the first qubit)
H(q[0])
# Add a CNOT gate with qubit 0 and qubit 1 being the control qubit and the target qubit, respectively
CX(q[0], q[1])

# Measure both qubits
MeasureZ(q, range(2))
# Measure the circuit for 2048 times and save the result in taskResult
taskResult = env.commit(2048, fetchMeasure=True)
pprint(taskResult)
```

Run the same code locally. It won't cost the Credit.

```
from pprint import pprint
import sys
sys.path.append('../..')
from QCompute import *

# Create a new quantum environment
env = QuantumEnvironment()
# Use the local quantum simulator developed by Baidu
env.backend(BackendName.LocalBaiduSim2)

# Initialize two qubits at state |00>
q = [env.Q[0], env.Q[1]]
# Add a Hadamard gate on qubit 0 (the first qubit)
H(q[0])
# Add a CNOT gate with qubit 0 and qubit 1 being the control qubit and the target qubit, respectively
CX(q[0], q[1])

# Measure both qubits
MeasureZ(q, range(2))
# Measure the circuit for 2048 times and save the result in taskResult
taskResult = env.commit(2048, fetchMeasure=True)
pprint(taskResult)
```

Using cloud servers, compared to running all code locally, has a longer time for result retrieval. However, cloud servers have much larger computing power than personal computers. Users are suggested to use cloud resources when they need to simulate dozens of qubits. Of course, there won't be any problem with running code locally when users are executing some simple small-scale tasks.

Above is a quick guide to QCompute. There are tutorials on some popular quantum algorithms in the folder. Users can learn and play with them at will.

3 Toolbox

Modify parameters

Print result every time or not .... Define/Settings/outputInfo
Print as binary or hexadecimal digit .... Define/Settings/measureFormat
Start Sim2 internally (faster) or externally (more stable) ....
OpenSimulator/local_baidu_sim2/commit
Check available backends .... QuantumPlatform/__init__/BackendName

FAQ

1. How to program on Quantum Leaf?
   (a) Users can use Python programming language and simple quantum programming language for experiments and developmen.
   (b) If users use PyOnline or QComposer on Quantum-hub to do experiments, the platform will invoke Token automatically and it is unnecessary for the users to fill in the token by themselves.
   (c) If users use QCompute to do experiments, the above codes listed as the example shall be used to transmit the Token, which will be used to connect to the remote simulator or device.
2. Why should you use QCompute?
   (a) Users will be able to transfer and use classic language knowledge.
   (b) Users will be able to run algorithms that will adjust subsequent computing according to current results (i.e. variation algorithm).
3. What is the difference between local simulator and on-line simulator?
   (a) Local simulator uses local resources, while on-line simulator uses computing resources from Baidu Cloud.
   (b) The computing power of on-line simulator is greater with the support of Baidu Cloud computing.
4. How to accelerate the computing process?
   (a) Use online simulator.
   (b) Concurrent computing method is recommended for interactive computing.
5. Uses of Credits
   (a) On-line resource computing will consume 1 credit at one time.
   (b) Please inform quantum@baidu.com after using any of the credits.
6. How to subscribe as a VIP?

   Users can subscribe as a VIP with an invitation after going through the following steps:
   (a) Verification of corresponding identity.
   (b) Submit VIP subscribing application to e-mail:quantum@baidu.com or feedback&suggestion in Quantum-hub.
   (c) The application would be approved if the back stage confirms.