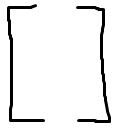
L3.1

6:04 If You have a 3 dimensional Hilbert space with l0>, l1> and l2>

How would You describe the wavefunction Psi like:

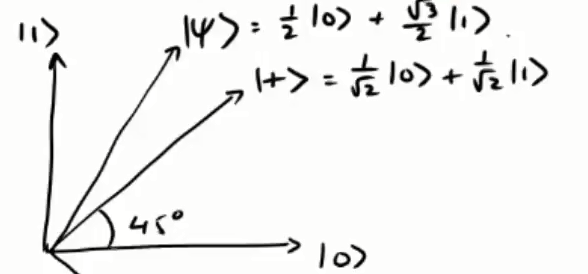
l**ψ> =**  **\_\_\_** l0> + **\_\_\_** l1> + **\_\_\_** l2>

How would You describe the wavefunction as a vector:

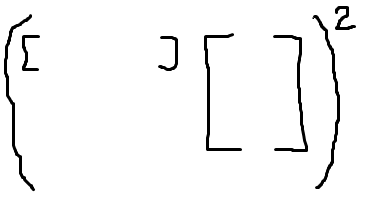


IF we have a K-dimensional Hilbert with 3 vectors what should the value of K be ? \_\_\_\_\_\_\_

What is the probability that the outcome is l+> “Inner product ” between l**ψ> and l+>** for the 2 dimensional Hilbert space below**?**

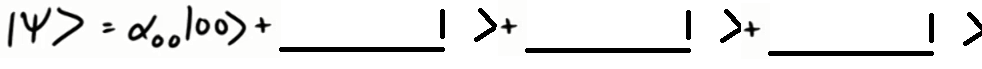


Please enter the numbers below into the Inner product below



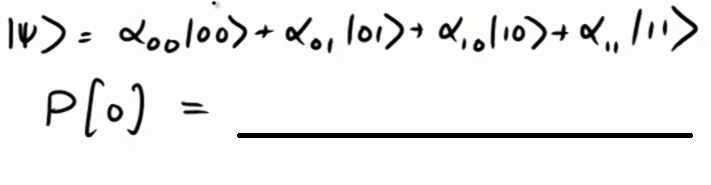
L3.2

0:54 How many possibilities are there – please fill out the rest below

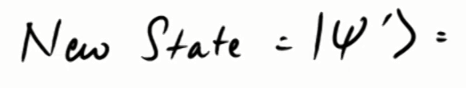


1:44 Which of the states above could the system with 2 Qubits be in ? :\_\_\_\_\_\_\_\_\_\_\_\_\_

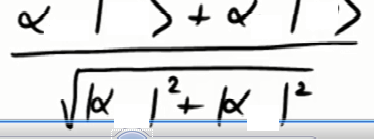
4:20 What is the probability that we see the first of the two qubits in state zero ?



What is the new state ?

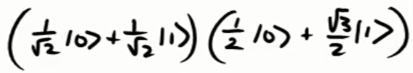
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What is the new state normalized ?



L3.3

1:36 What would be the outcome of multiplying Q1\*Q2 shown below – Pleasewirite the answer on the below line?

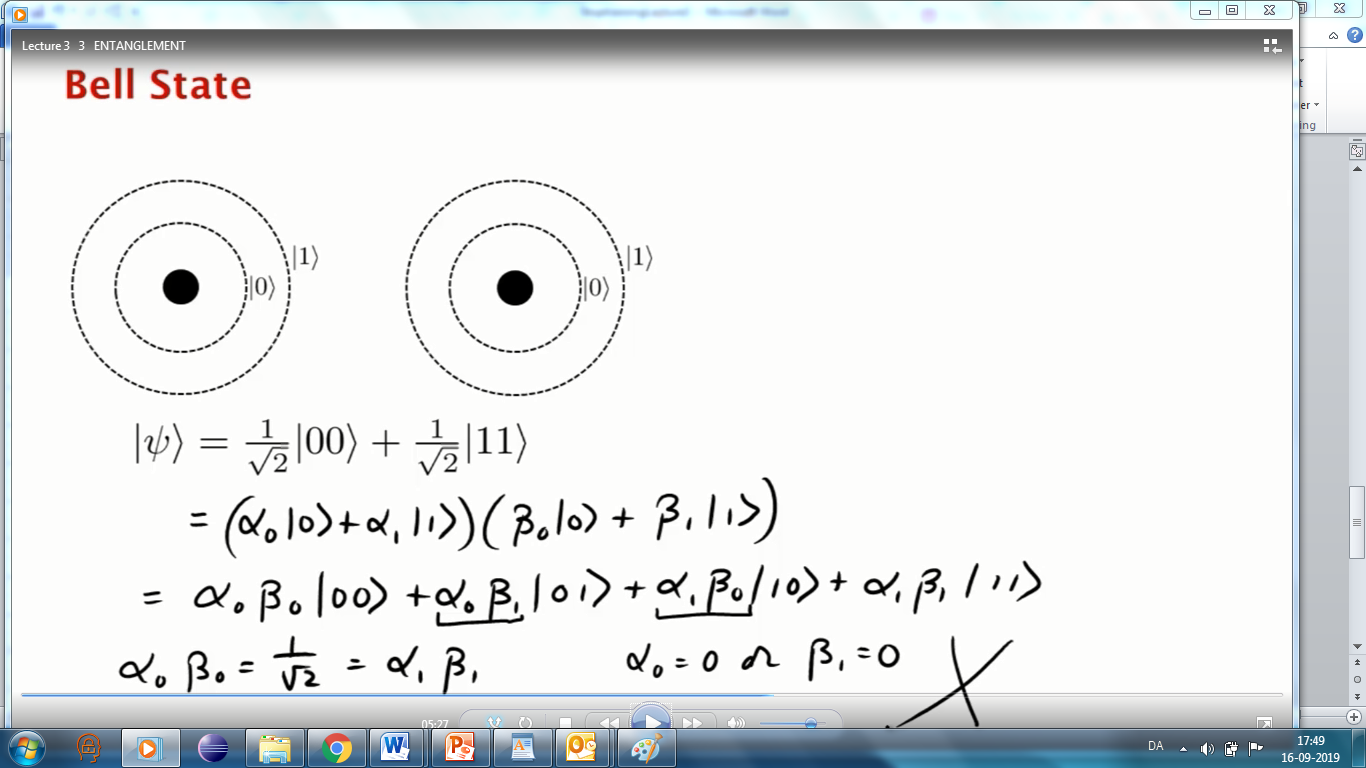


Result A

If only the Result A was given could You then in a reversible way find out what the original states of the two qubits was ? \_\_\_\_\_\_\_\_\_\_\_

Why Not ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Please explain below



8:57 What do You call this mysterious property ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

L3.4 Who invented the EPR Paradox\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0:59 How is a Bell state written ? l**ψ> =**  **\_\_\_** l\_ \_> + **\_\_\_** l\_ \_>

7:02 What is the difference between the Bit and the sign value ?

8:56 Why did E,P and R concluded that the Quantum Mechanics was an incomplete theory ?

How many Years wasted Einstein of his life to undermine ***Laws of Quantum mechanics*** ? \_\_\_\_\_\_\_\_\_\_\_

# Exercises

## HelloWorld

Something with sss.measure(x,y)

[IBM **Developer**](https://developer.ibm.com/?lnk=m)

Øverst på formularen

Submit

Nederst på formularen

* My IBM
* Site navigation
* [IBM Code](https://developer.ibm.com/code/)
* [Code](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/)
* [Content](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/)
* [Community](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/)
* [Open Source](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/)

[Open](https://developer.ibm.com/code/category/open/)[QISKit](https://developer.ibm.com/code/category/qiskit/)

# A developer’s guide to using the Quantum QISKit SDK

[Ismael Faro](https://www.linkedin.com/in/ismaelfaro/)  
Published on May 17, 2017*/ Updated on May 22, 2017*

[9](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comments)

[QISKit (Quantum Information Software Kit)](https://developer.ibm.com/open/openprojects/qiskit/) is a Python software development kit (SDK) that you can use to create your quantum programs, based in circuits over [OpenQASM 2.0](https://github.com/IBM/qiskit-openqasm) Specs, compile and execute the SDK on several backends (via our upgraded 16 qubit chip, as well as online and local simulators). For the online backend QISKit, use our [python API connector](https://github.com/IBM/qiskit-api-py) to the [IBM Quantum Experience](http://quantumexperience.ng.bluemix.net/?cm_sp=dw-bluemix-_-code-_-devcenter).

In this tutorial, learn how to use the QISKit SDK, from a developer’s point of view. We will show you how to install and start to use the SDK tools.

## Install QISKit

The easiest way to install QISKit is to use the Anaconda Python distribution. [Visit the Anaconda site](https://www.continuum.io/downloads) for information on how to install Anaconda.

After you’ve installed Anaconda, install your QISKit from the git repository.

* Clone the repo:

git clone https://github.com/IBM/qiskit-sdk-py.git

cd qiskit-sdk-py-dev

* Create the environment with the following dependencies: make env

## Use the quantum examples

You can use the examples in an easy way with Jupyter or Python.

* Add your API token to the file "Qconfig.py". Get the file from the [IBM Quantum Experience](https://quantumexperience.ng.bluemix.net/?cm_sp=dw-bluemix-_-code-_-devcenter) account: cp tutorial/Qconfig.py.default Qconfig.py
* Run the Jupyter notebook by using the command make run

## Put QISKit to use

### Step 1. Create a program

First, you need to import the QuantumProgram from QISKit

**import sys**

sys.path.append("../../") # solve the relative dependencies if you clone QISKit from the Git repo and use like a global.

**from qiskit import** QuantumProgram

**import Qconfig**

The basic elements that you need to create your first program are the QuantumProgram, one circuit, one quantum register, and one classical register.

# Creating Programs

# create your first QuantumProgram object instance. Q\_program =

QuantumProgram()

# Creating Registers

# create your first Quantum Register called "qr" with 2 Qbit qr =

Q\_program.create\_quantum\_registers("qr", 2)

 # create your first Classical Register called "cr" with 2 bit cr =

Q\_program.create\_classical\_registers("cr", 2)

# Creating Circuits

 # create your first Quantum Circuit called "qc" related with your Quantum Regis ter "qr"

 # and your Classical Register "cr"

qc = Q\_program.create\_circuit("qc", ["qr"], ["cr"])

**Output:**  
   >> quantum\_registers created: qr 2  
   >> classical\_registers created: cr 2

Another option for creating your QuantumProgram instance is to define a dictionary that includes all the components of your program.

Q\_SPECS = {

"name": "Program-tutorial",

"circuits": [{

"name": "Circuit",

"quantum\_registers": [{

"name":"qr",

"size": 4

}],

"classical\_registers": [{

"name":"cr",

"size": 4

}]}],

}

One program must have a name and one circuit array. Any circuit must have a name, and it can have several quantum registers and several classical registers. Every register needs to have a name and the number of the elements (Qbits or bits).

After that, you can use this dictionary definition like the specs of one QuantumProgram object to initialize it.

Q\_program = QuantumProgram(specs=Q\_SPECS)

**Output**  
   >> quantum\_registers created: qr 4  
   >> classical\_registers created: cr 4

You can get every component from you new\_Q\_program to use.

#get the components.

# get the circuit by Name

circuit = Q\_program.get\_circuit("Circuit")

# get the Quantum Register by Name

quantum\_r = Q\_program.get\_quantum\_registers("qr")

# get the Classical Register by Name

classical\_r = Q\_program.get\_classical\_registers('cr')

### Step 2. Add gates to your circuit

After you create the circuit with its registers, you can add gates to manipulate the registers. The following listing shows you all the gates that you can use.

You can find extensive information about these gates and how use them in our [Quantum Experience User Guide](https://quantumexperience.ng.bluemix.net/qstage/#/tutorial?sectionId=71972f437b08e12d1f465a8857f4514c&%20pageIndex=2?cm_sp=dw-bluemix-_-code-_-devcenter).

# H (Hadamard) gate to the Qbit 0 in the Quantum Register "qr"

circuit.h(quantum\_r[0])

# Pauli X gate to the Qbit 1 in the Quantum Register "qr"

circuit.x(quantum\_r[1])

# Pauli Y gate to the Qbit 2 in the Quantum Register "qr"

circuit.y(quantum\_r[2])

# Pauli Z gate to the Qbit 3 in the Quantum Register "qr"

circuit.z(quantum\_r[3])

# Cnot (Controlled-NOT)gate from Qbit 0 to the Qbit 3

circuit.cx(quantum\_r[0], quantum\_r[3])

# add a barrier to your circuit

circuit.barrier()

# first physical gate: u1(lambda) to Qbit 0

circuit.u1(0.3, quantum\_r[0])

# second physical gate: u2(phi,lambda) to Qbit 1

circuit.u2(0.3, 0.2, quantum\_r[1])

# second physical gate: u3(theta,phi,lambda) to Qbit 2

circuit.u3(0.3, 0.2, 0.1, quantum\_r[2])

# S Phase gate to Qbit 0

circuit.s(quantum\_r[0])

# T Phase gate to Qbit 1

circuit.t(quantum\_r[1])

# identity gate to Qbit 1

circuit.iden(quantum\_r[1])

# Classical if, from Qbit2 gate Z to classical bit 1

circuit.z(quantum\_r[2]).c\_if(classical\_r, 1)

# measure gate from the Qbit 0 to Classical bit 0

circuit.measure(quantum\_r[0], classical\_r[0])

### Step 3. Extract QASM

You can obtain a QASM representation of your code.

**Input:**

# QASM from a program

QASM\_source = Q\_program.get\_qasm("Circuit")

print(QASM\_source)

**Output:**

OPENQASM 2.0;

include "qelib1.inc";

qreg qr[4];

creg cr[4];

h qr[0];

x qr[1];

y qr[2];

z qr[3];

cx qr[0],qr[3];

barrier qr[0],qr[1],qr[2],qr[3];

u1(0.300000000000000) qr[0];

u2(0.300000000000000,0.200000000000000) qr[1];

u3(0.300000000000000,0.200000000000000,0.100000000000000) qr[2];

u1(1.570796326794897) qr[0];

u1(0.785398163397448) qr[1];

id qr[1];

if(cr==1) z qr[2];

measure qr[0] -> cr[0];

Notice that both codes don't have the same notation. The second one returns a conversion from every H, X, Y, Z gate the the u1, u2 and u3 represents.

### Step 4. Compile and run or execute

**Input**

device = 'simulator' #Backed where execute your program, in this case in the on line simulator

circuits = ['Circuit] #Group of circuits to exec

Q\_program.set\_api(Qconfig.APItoken, Qconfig.config["url"])

#set the APIToken and API url

**Output:** True

**Input:**

Q\_program.compile(circuits, device) # Compile your program

result = Q\_program.run(wait=2, timeout=240) # Run your

program in the device and check the execution result

every 2 seconds

print(result)

print(Q\_program.get\_qasms(circuits)) (circuits))

**Output**

{'qasm': 'OPENQASM 2.0;\ninclude "qelib1.inc";\nqreg qr[4];\ncreg cr[4];\nu1(3

.141592653589793) qr[3];\nu3(3.141592653589793,1.5707963267948966,1.5707963267

948966) qr[2];\nu3(3.141592653589793,0.0,3.141592653589793) qr[1];\nu2(0.0,3.1

41592653589793) qr[0];\ncx qr[0],qr[3];\nbarrier qr[0],qr[1],qr[2],qr[3];\nu1(

0.3) qr[0];\nu1(1.570796326794897) qr[0];\nu2(0.3,0.2) qr[1];\nu1(0.7853981633

97448) qr[1];\nid qr[1];\nu3(0.3,0.2,0.1) qr[2];\nif(cr==1) u1(3.1415926535897

93) qr[2];\nmeasure qr[0] -> cr[0];\n'}

status = RUNNING (2 seconds)

status = RUNNING (4 seconds)

status = RUNNING (6 seconds)

{'qasms': [{'qasm': '\ninclude "qelib1.inc";\nqreg qr[4];\ncreg cr[4];\nu1(3.1

41592653589793) qr[3];\nu3(3.141592653589793,1.5707963267948966,1.570796326794

8966) qr[2];\nu3(3.141592653589793,0.0,3.141592653589793) qr[1];\nu2(0.0,3.141

592653589793) qr[0];\ncx qr[0],qr[3];\nbarrier qr[0],qr[1],qr[2],qr[3];\nu1(0.

3) qr[0];\nu1(1.570796326794897) qr[0];\nu2(0.3,0.2) qr[1];\nu1(0.785398163397

448) qr[1];\nid qr[1];\nu3(0.3,0.2,0.1) qr[2];\nif(cr==1) u1(3.141592653589793

) qr[2];\nmeasure qr[0] -> cr[0];\n', 'status': 'DONE', 'executionId': '62c466

7cf29a35ad009e8754baa85c80', 'result': {'date': '2017-05-14T08:45:36.552Z', 'd

ata': {'time': 0.2707841396331787, 'counts': {'0000': 516, '0001': 508}}}}], '

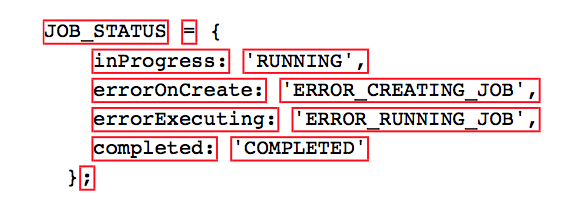
shots': 1024, 'backend': {'name': 'simulator'}, 'status': 'COMPLETED', 'maxCre

dits': 3, 'usedCredits': 0, 'creationDate': '2017-05-14T08:45:35.262Z', 'delet

ed': False, 'id': '451903c7de63febb2614adbdeeaeb485', 'userId': '933a590a13327

c52eb5310484d6bcf54'}

When you run a program, the result can be:



The run() command waits until timeout or when it receives the "COMPLETED" or some error message.

You can use the execute() command to make the compile and run command happen in a unique step.

**Input**

result = Q\_program.execute(circuits, device, wait=2, timeout=240)

print(Q\_program.get\_qasms(circuits))

**Output:**

{'qasm': 'OPENQASM 2.0;\ninclude "qelib1.inc";\nqreg qr[4];\ncreg cr[4];\nu1(3

.141592653589793) qr[3];\nu3(3.141592653589793,1.5707963267948966,1.5707963267

948966) qr[2];\nu3(3.141592653589793,0.0,3.141592653589793) qr[1];\nu2(0.0,3.1

41592653589793) qr[0];\ncx qr[0],qr[3];\nbarrier qr[0],qr[1],qr[2],qr[3];\nu1(

0.3) qr[0];\nu1(1.570796326794897) qr[0];\nu2(0.3,0.2) qr[1];\nu1(0.7853981633

97448) qr[1];\nid qr[1];\nu3(0.3,0.2,0.1) qr[2];\nif(cr==1) u1(3.1415926535897

93) qr[2];\nmeasure qr[0] -> cr[0];\n'}

status = RUNNING (2 seconds)

status = RUNNING (4 seconds)

status = RUNNING (6 seconds)

#### Compile parameters

The quantum compile parameters are: Q\_program.compile(circuits, device="simulator", shots=1024, max\_credits=3, basis\_gates=None, coupling\_map=None)

* circuits: Array of circuit to compile
* device: Backend  
  ["ibmqx\_qasm\_simulator", #Online simulator  
  "ibmqx2", #Online RealChip, 5Qbits  
  "ibmqx3:, #Online RealChip, 16Qbits  
  "local\_unitary\_simulator", #Local unitary Simulator  
  "local\_qasm\_simulator"] # Local Simulator
* shots: Number of shots, only for real chips
* max\_credits: Maximum number of the credits to spend in the executions. If the executions are more expensive, the job is aborted, only the real chips
* basis\_gates: are the base gates. By default, they are: u1,u2,u3,cx,id
* coupling\_map: Object that represent the physical/topological Layout in a chip.

#### Run parameters

The run parameters are: Q\_program.run(wait=5, timeout=60)

* wait: Time to wait to check if the execution is COMPLETED.
* timeout: Timeout of the execution.

#### Execute parameters

The execute command has the combined parameters of Compile and Run:

Q\_program.execute(circuits, device, shots=1024, max\_credits=3, basis\_gates=None, wait=5, timeout=60, basis\_gates=None, coupling\_map=None,)

### Step 5. Execute in a real chip

**Input:**

device = 'qx5qv2' #Backed where execute your program, in this case in the Real Quantum Chip online

circuits = [circuit] #Group of circuits to exec

shots = 1024m #Number of shots to run the program (experiment), Maximum 8192 shots.

max\_credits=3s. #Maximum number of the credits to spend in the executions

result = Q\_program.execute(circuits, device, shots, max\_credits=3, wait+10, timeout=240)

print(Q\_program.get\_qasms(circuits))

**Output:**

{'qasm': 'OPENQASM 2.0;\ninclude "qelib1.inc";\nqreg qr[4];\ncreg cr[4];\nu1(3

.141592653589793) qr[3];\nu3(3.141592653589793,1.5707963267948966,1.5707963267

948966) qr[2];\nu3(3.141592653589793,0.0,3.141592653589793) qr[1];\nu2(0.0,3.1

41592653589793) qr[0];\ncx qr[0],qr[3];\nbarrier qr[0],qr[1],qr[2],qr[3];\nu1(

0.3) qr[0];\nu1(1.570796326794897) qr[0];\nu2(0.3,0.2) qr[1];\nu1(0.7853981633

97448) qr[1];\nid qr[1];\nu3(0.3,0.2,0.1) qr[2];\nif(cr==1) u1(3.1415926535897

93) qr[2];\nmeasure qr[0] -> cr[0];\n'}

### Result

You can access the result using get\_counts:

**Input:** Q\_program.get\_counts("Circuit")  
**Output:** {'0000': 516, '0001': 508}

Or you can use the function tool get\_data(n) to obtain it directly:

**Input:** Q\_program.get\_data(result,0)  
**Output:** {'counts': {'00000': 532, '00001': 492}, 'time': 14.103801012039185}

## Conclusion

Want more information about the IBM Quantum Experience? We have other tutorials that introduce you to more complex concepts directly related to the upgraded service. For more information about how to use the Quantum Experience, consult the [tutorials](https://quantumexperience.ng.bluemix.net/qstage/#/tutorial?sectionId=c59b3710b928891a1420190148a72cce&pageIndex=0?cm_sp=dw-bluemix-_-code-_-devcenter) on IBM Bluemix or check out [the community](https://quantumexperience.ng.bluemix.net/qstage/#/community?cm_sp=dw-bluemix-_-code-_-devcenter).

**TAGS**[**QISKIT**](https://developer.ibm.com/code/tag/qiskit/)**,**[**QUANTUM COMPUTING**](https://developer.ibm.com/code/tag/quantum-computing/)

9 comments on"A developer’s guide to using the Quantum QISKit SDK"

1. [**IBM doubles compute power for quantum systems, developers execute 300K+ experiments on IBM Quantum Cloud - The developerWorks Blog**](https://developer.ibm.com/dwblog/2017/quantum-computing-16-qubit-processor/) [May 17, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-209)

[…] QISKit – Python-based SDK for the Quantum Experience […]

1. [**IBM Unveils Its Most Powerful Quantum Processor Yet for Business and Science – Technology Up2date**](http://observatorios.eversusi.com/tecnologia/ibm-unveils-its-most-powerful-quantum-processor-yet-for-business-and-science/) [May 20, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-212)

[…] InfoQ reported, IBM provides a Python-based quantum development SDK called QISKit that can be used to run experiments on the IBM Q processors. IBM’s new 16 qubit […]

1. [**A beginner’s guide to quantum computing - developerWorks TV**](https://developer.ibm.com/tv/beginners-guide-quantum-computing/) [June 12, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-227)

[…] Explore the QISKit, an SDK and API for Q computing; it comes with a developer’s guide to using the QISKit […]

1. **jose contreras** [November 07, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-697)

I am behind a proxy and i can’t succesfully execute PIP INSTALL QISKIT command. ¿is there any other method to install QISKIT? or a workarround to download an QISKIT offline istaller.  
thanks in advance for your attention

1. **ying** [November 30, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-787)

I have some questions about the qiskit API, I want to know how to init the register, for example, I set 2 quantum\_register, I want to init them with |1>,|0> , how should I do? need your help, thx!

1. **ismaelfarosertage** [December 05, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-819)

Hi Jose,

You can download the QISKit directly from the Github page <https://github.com/QISKit/qiskit-sdk-py> and you can follow the instructions to use it directly.

1. **ismaelfarosertage** [December 05, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-820)

by default all the qubits are initialized to state 0, you need to use a X gate to Flip the state, you can see more information about it in our user guides, [https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=beginners-guide&page=005-Single-Qubit\_Gates~2F001-Single-Qubit\_Gates](https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=beginners-guide&page=005-Single-Qubit_Gates~2F001-Single-Qubit_Gates&cm_sp=dw-bluemix-_-code-_-devcenter)

1. **Manuel** [December 30, 2017](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-909)

Hi Ismael!

I was trying to execute a 5 credits script using 8192 shots in the ibmqx2 but I am missing something apparently, because it runs perfect at 1024 shots with this line:

result = Q\_program.execute([“circuit”], backend=’ibmqx2′, shots=1024)

but later, in when I change it to:

result = Q\_program.execute([“superposition”], backend=’ibmqx2′, shots=8192 , max\_credits=5 )

I get some traceback error in \_quantumprogram.py and using result.get\_error() it says is a ‘Time out’ error  Any idea?

Thanks!  
Cheers

1. [**Exploring the SDK for Quantum Computing - The developerWorks Blog**](https://developer.ibm.com/dwblog/2018/exploring-sdk-quantum-computing/) [January 31, 2018](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/#comment-1046)

[…] QISKit – Python-based SDK for the Quantum Experience […]

### Join The Discussion

Øverst på formularen

Your email address will not be published. Required fields are marked \*



Name \* 

Email \*

Website



Nederst på formularen

* [IBM **Developer**](https://developer.ibm.com/)
* [About](https://developer.ibm.com/about)
* [Site Feedback & FAQ](https://developer.ibm.com/feedback)
* [Submit content](https://developer.ibm.com/terms/submission-of-content-agreement/)
* [Report abuse](https://www.ibm.com/developerworks/secure/report/)
* [Third-party notice](https://developer.ibm.com/terms/third-party-notice/)

##### **Follow us**

[Twitter](https://twitter.com/IBMDeveloper)

[Facebook](https://www.facebook.com/IBMDeveloper)

[LinkedIn](https://www.linkedin.com/showcase/developerworks/)

[YouTube](https://www.youtube.com/channel/UCUm6InQvGI9-6vo1teGWINA)

* **Select a language**
* [English](https://developer.ibm.com/)
* [中文](https://www.ibm.com/developerworks/cn/)
* [日本語](https://www.ibm.com/developerworks/jp/)
* [Русский](https://www.ibm.com/developerworks/ru/)
* [Português](https://www.ibm.com/developerworks/br/)
* [Español](https://www.ibm.com/developerworks/ssa/)
* [한글](https://developer.ibm.com/kr/)
* [Code Patterns](https://developer.ibm.com/patterns/)
* [Articles](https://developer.ibm.com/articles/)
* [Tutorials](https://developer.ibm.com/tutorials/)
* [Recipes](https://developer.ibm.com/recipes/)
* [Open Source Projects](https://developer.ibm.com/open/)
* [Videos](https://developer.ibm.com/videos/)
* [Newsletters](https://developer.ibm.com/newsletters/)
* [Events](https://developer.ibm.com/events/)
* [Cities](https://developer.ibm.com/cities/)
* [Developer Answers](https://developer.ibm.com/answers/)
* [Contact IBM](https://www.ibm.com/contact/us/en/?lnk=flg-cont-usen)
* [Privacy](https://www.ibm.com/privacy/us/en/?lnk=flg-priv-usen)
* [Terms of use](https://www.ibm.com/us-en/legal?lnk=flg-tous-usen)
* [Accessibility](https://www.ibm.com/accessibility/us/en/?lnk=flg-acce-usen)
* [Feedback](https://developer.ibm.com/code/2017/05/17/developers-guide-to-quantum-qiskit-sdk/)

United States - English