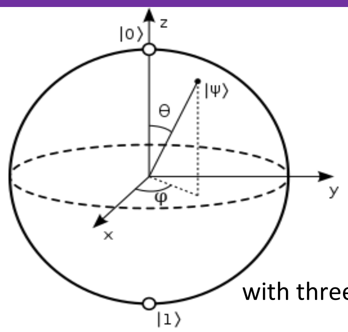
IBM QISKit cheat sheet

qubits



$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \ |\alpha|^2 + |\beta|^2 = 1$$

- For any possible state: the measurement can only result in $|0\rangle$ or $|1\rangle$
- Probability of measuring $|0\rangle$ is $|\alpha|^2$, probability of measuring $|1\rangle$ is $|\beta|^2$
- When the measurement is done, superposition is lost and the qubit is set in the state just measured.

with two qubits : $|\phi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$

with three qubits : $|\phi\rangle = a|000\rangle + b|001\rangle + c|010\rangle + d|011\rangle + e|100\rangle + f|101\rangle + g|110\rangle + h|111\rangle$ and so on...

operators (gates)

« PAULI » Operators

rotation around the x axis	X	$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$	$qc.x(qr[n])$	R_x	$\begin{pmatrix}\cos\frac{\theta}{2} & -i\sin\frac{\theta}{2}\\-i\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}$
rotation around the y axis	Y	$\begin{pmatrix}0 & -i\\i & 0\end{pmatrix}$	$qc.y(qr[n])$	R_y	$\begin{pmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}$
rotation around the z axis	Z	$\begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}$	$qc.z(qr[n])$	R_z	$\begin{pmatrix}e^{-i\frac{\theta}{2}} & 0\\0 & e^{i\frac{\theta}{2}}\end{pmatrix}$
Identity (do nothing)	Id	$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	$qc.id(qr[n])$		

$$X^2 = Y^2 = Z^2 = I$$

$$XY = iZ \,;\, ZX = iY \,;\, YZ = iX$$

$$XY = -YX \,;\, YZ = -ZY;\, XZ = -ZX$$

more operators are availablefrom qiskit (swap, cswap, ccnot, cz, ...)

U_1	$\begin{pmatrix}1 & 0\\0 & e^{i\theta}\end{pmatrix}$	U1 gate is known as the phase gate and is essentially the same as Rz(θ).
U_2	$\begin{pmatrix}1 & -e^{i\lambda}\\e^{i\phi} & e^{i\lambda+i\phi}\end{pmatrix}$	From this gate, the Hadamard is done by H=U2(0,π)
U_3	$\begin{pmatrix}\cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2}\\e^{i\phi}\sin\frac{\theta}{2} & e^{i\lambda+i\phi}\cos\frac{\theta}{2}\end{pmatrix}$	U3 has the effect of rotating a qubit to a state with an arbitrary superposition and relative phase

S — $\sqrt{Z}$ phase gate X → Y

$S^\dagger$ — phase gate X → -Y

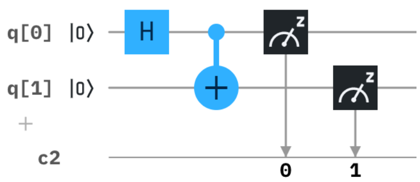
T — $\sqrt{S}$ phase gate

superposition (X+Z) Hadamard gate — H — $\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$ — $qc.h(qr[n])$

measurement — from quantum state in quantum register to 0 or 1 in classical register

CNOT : flips target qubit depending on control qubit state.

circuits



Circuits are using quantum bits (starting in state |0> grouped in quantum registers), **classical register** for measurement reading**, and gates applied to qubits from left to right in time sequence.**

IBM Q Experience : https://www.ibm.com/quantum-computing/

IBM portal contains documentation, examples, workbooks. Build quantum circuits using a graphical composer and execute on real quantum device or online simulator for free. Also provides API key for accessing available IBM Q Systems.

Circuit Composer

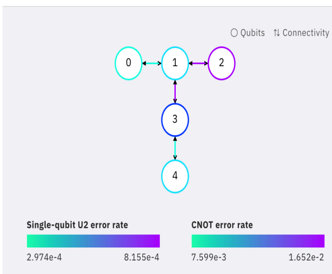
Explore the graphical interface for creating and testing circuits

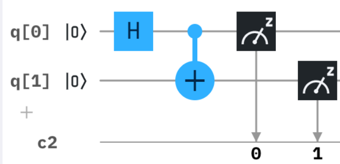
Create a circuit →

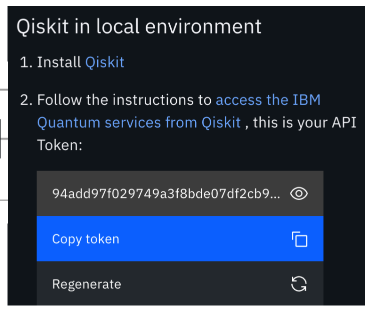
Qiskit Notebooks

Create your first notebook and start using Qiskit

Create a notebook →







conda (for working with Jupyter Notebooks)

Donwload and install anaconda (@ anaconda.org)
Open a terminal (or conda terminal in Windows®).
Some usefull conda commands :
(**qc**, **qc2** being example for « environment » names)

- `conda create -n qc python=3.X`
- `conda create --clone qc -n qc2`
- `conda activate qc2`
- `conda env remove -n qc`
- `conda env list` (envs) `conda list` (packages)
- `conda env update`
- `pip install qiskit`
- `pip install qiskit --upgrade`

Activate and launch: (from terminal) :

- `conda activate qc`
- `jupyter notebook`

QISKit cheat-sheet

qiskit (Python 3)

```
import qiskit
qiskit.__qiskit_version__                     # (qiskit.__version__ returns terra's version)

{'qiskit-terra': '0.11.1',                    # building, compiling and executing circuits
 'qiskit-aer': '0.3.4',                       # working with simulators
 'qiskit-ignis': '0.2.0',                     # understanding and mitigating noise
 'qiskit-ibmq-provider': '0.4.5',             # accessing IBM backends
 'qiskit-aqua': '0.6.2',                      # library of quantum computing applications
 'qiskit': '0.14.1'}                          # main module
```

anatomy of « Hello World! » quantum program

```
import qiskit                                 # import qiskit module
q = QuantumRegister(2)                        # define a quantum register for 2 qubits
c = ClassicalRegister(2)                      # define a classical register for 2 bits
qc = QuantumCircuit(q,c)                      # define a quantum curcuit using q and c
qc.h(0)                                       # apply Hadamard gate on qubit 0
qc.cx(0,1)                                    # apply CNOT from qubit 0 to qubit 1 as target.
qc.measure(q,c)                               # measure states of qubits in q into register c
qc.draw(output='mpl')                         # visualize circuit with matplotlib rendering
backend = qiskit.Aer.get_backend('qasm_simulator')  #select backend (local simulator)
job = qiskit.execute(qc,backend,shots=1000)   # execute circuit qc on selected backend, 1000 times
result = job.result()                         # fetch job results.
print(result.get_counts(qc))                  # gets result count on basis states into a python dict
{'00': 504, '11': 496}.                       # maybe
```

qiskit IBM Q Provider

```
IBMQ.save_account('**my token**',overwrite=True)# one time setup (saving token locally)
IBMQ.stored_account()                         # retrieve account from you environement
IBMQ.load_account()                           # enable account
sel_prov = IBMQ.get_provider(hub='ibm-q')     # select provider (you may have many, see IBMQ.providers)
print(sel_prov.backends())                    # list available backends
backend = sel_prov.get_backend('ibmqx2')      # select one of the available backends
backend.configuration()                       # backend details: qubits count, coupling map, gate config…
backend.status()                              # current status and pending jobs count
job.job.id                                    # fetch id to enable results retrieval in case of long wait
tools.monitor.job_monitor(job)                # monitoring my job status in queue
```

qiskit terra

```
circ.to_instruction()                         # transform a circuit into a single instruction
qc.append(circ, <input qubits>)               # add circ as a single gate to quantum circuit qc
```

transpilation and optimization (with: `from qiskit.complier import transpile` :

```
circ= transpile(qc,backend=backend,optimization_level=N) #N=0 : mapping only, N=1 gates cancellation,
                                              # N=2 noise adaptive layout + gate cancell based on
                                    # commutation relationship, N=3 resynthesis of 2-qubits blocks
```

qiskit aer

provides state vector simulator (on top of qasm_simulator which simulates a physical device) :

```
backend = Aer.get_backend('statevector_simulator')
input_state = [1/sqrt(2),  1j/sqrt(2) ]       # arbitrary initial state can be loaded
execute(circ, backend, backend_options={"initial_statevector": input_state})
result().get_statevector(qc)                  # provides the quantum state of the system (ie: state
                                              # vector coordinates)
```

can also be used to simulate with noise (using self defined or provided noise models).

qiskit ignis

This workbook can be run from the IBM site and provides examples for how to use the `ignis.mitigation.measurement` module:
https://quantum-computing.ibm.com/jupyter/tutorial/advanced/ignis/4_measurement_error_mitigation.ipynb

qiskit aqua

```
dir(aqua.algorithms)                          # (from qiskit import *) lists available AQUA algorithms
```

specify algorithm, parameters, and backend in a JSON formatted variable (named params in this case) then:

```
result = run_algorithm(params, backend=backend)# qiskit builds and run the circuit as defined in params
```

open pulse

```
dir(qiskit.pulse)                             # provides the list of available functions
```