

Builins and Primitives

Gate	Q#	ProjectQ	Cirq	Qiskit	PyQuil
<i>I</i>	I			iden	I
<i>H</i>	H	H	H	h	H
<i>S</i>	S	S	S	s	S
<i>T</i>	T	T	T	t	T
<i>X, NOT</i>	X	X	X	x	X
<i>Y</i>	Y	Y	Y	y	Y
<i>Z</i>	Z	Z	Z	z	Z
<i>R_x</i>	Rx	Rx	RotXGate	rx	Rx
<i>R_y</i>	Ry	Ry	RotYGate	ry	Ry
<i>R_z</i>	Rz	Rz	RotZGate	rz	Rz
<i>R_φ</i>	R1	R		u_1	PHASE
Measure	M	Measure	measure	measure	MEASURE
Barrier		Barrier		barrier	
CX, CNOT	CNOT	CNOT	CNOT	cx	CNOT
CCX, CCNOT, Toffoli	CCNOT	Toffoli	CCX, TOFFOLI	ccx	CCNOT
SWAP	SWAP	Swap	SwapGate	swap	SWAP
CZ	(Controlled Z)	CZ	CZ	cz	CZ
CSWAP, Fredkin	(Controlled SWAP)	C(Swap)	CSWAP, FREDKIN	cswap	CSWAP
CR _z	(Controlled Rz)	CRz		crz	CPHASE
ISWAP			ISWAP		ISWAP
QFT	QFT	QFT			
Other	HY, RAll0, RAll1	Sdag, Tdag, SqrtX, SqrtSwap, Entangle, TimeEvolution, QubitOperator, PhaseOracle, PermutationOracle, AddConstantModN	Rot11Gate, CCZ	cy, ch, rzz	PSWAP, CPHASE00, CPHASE01, CPHASE10

Operation	Q#	ProjectQ	Cirq	Qiskit	PyQuil
Gate from matrix		<code>G = BasicGate()</code> <code>G.matrix = numpy.matrix</code>	<code>SingleQubitMatrixGate,</code> <code>TwoQubitMatrixGate</code>		<code>defgate</code>
Controlled	<code>(Controlled G)(c, q)</code>	<code>C(G) q # or</code> with <code>Control(eng, c):</code> <code>G q</code>	<code>ControlledGate(G)</code>	<code>G.q_if(q)</code>	
Inverse	<code>(Adjoint G)(q)</code>	with <code>Dagger(eng):</code> <code>G q</code>	<code>G.inverse()</code>	<code>G.inverse()</code>	<code>Program(G(q)).dagger()</code>
Apply to many qubits	<code>ApplyToEach(G, qs)</code>	<code>All(G) qs # or</code> <code>Tensor(G) qs</code>	<code>G.on_each(qs)</code>	<code>G(qs)</code>	
Simulators	local	local & cloud	local	local & cloud	cloud only
Execute on real quantum computer	no	IBM	Google	IBM	Rigetti
Rotation units	radians	radians	half-turns, radians, degrees	radians	radians
Integrations		Fermilib, OpenFermion	OpenFermion, QAsm	Qiskit-Aqua, QAsm	OpenFermion

```

1 // Q##
2 // Circuit.qs
3 namespace MyProgram {
4     open Microsoft.Quantum.Primitive;
5     open Microsoft.Quantum.Canon;
6
7     operation Circuit(q : Qubit) : (Result) {
8         body {
9             H(q);
10            let result = M(q);
11            return result;
12        }
13        // adjoint auto;
14        // controlled auto;
15        // controlled adjoint auto;
16    }
17 }

```

```

1 // Q##
2 // Driver.cs
3 using Microsoft.Quantum.Simulation.Core;
4 using Microsoft.Quantum.Simulation.Simulators;
5
6 namespace MyProgram {
7     class Driver {
8         static void Main(string[] args) {
9             using (var sim = new QuantumSimulator()) {
10                var result = Circuit.run(sim, 1, Result.Zero).Result;
11                System.Console.WriteLine($"Measured: {result}");
12            }
13        }
14    }
15 }

```

[1] Ryan LaRose. "Overview and Comparison of Gate Level Quantum Software Platforms". In: (July 6, 2018). arXiv: 1807.02500 [quant-ph]. URL: <http://arxiv.org/abs/1807.02500> (visited on 09/12/2018).

```

1  ## ProjectQ
2  from projectq import MainEngine
3  from projectq.ops import *
4
5
6  eng = MainEngine()
7  q = eng.allocate_qubit()
8
9
10 H          | q
11 Measure   | q
12
13 eng.flush()
14
15
16 print("Measured: {}".format(
17     int(q)
18 ))

```

```

1  ## Qiskit
2  from qiskit import *
3
4
5
6  q = QuantumRegister(1)
7  c = ClassicalRegister(1)
8  circuit = QuantumCircuit(q, c)
9
10 circuit.h(q)
11 circuit.measure(q, c)
12
13 sim = execute(circuit, "local_qasm_simulator", shots = 1)
14 result = sim.result()
15
16 print("Measured: {}".format(
17     list(result.get_counts())[0]
18 ))

```

```

1  ## Cirq
2  from cirq import *
3
4
5
6  q = GridQubit(0, 0)
7  circuit = Circuit()
8
9
10 circuit.append([H(q)])
11 circuit.append([measure(q, key = "c")])
12
13 sim = google.XmonSimulator()
14 result = sim.run(circuit)
15
16 print("Measured: {}".format(
17     int(result.measurements["c"][0,0])
18 ))

```

```

1  ## PyQuil
2  from pyquil.quil import Program
3  from pyquil.gates import *
4  from pyquil.api import QVMConnection
5
6  program = Program()
7
8
9
10 program.inst(H(0))
11 program.inst(MEASURE(0, 0))
12
13 qvm = QVMConnection()
14 result = qvm.run(program, [0])
15
16 print(result)

```