È bene segnalare che i codici di seguito rispecchiano i circuiti visti in esempio ma che possono variare in qualche porzione specifica. Ad esempio i backends utilizzati e il numero di shots lanciati. Questo per minimizzare le possibilità di errore quando i codici sono stati scritti.

```
#HALF ADDER
from qiskit import *
from qiskit import IBMQ
from qiskit.visualization import plot_histogram
get_ipython().run_line_magic('config', "InlineBackend.figure_format =
'svg' # Makes the images look nice")

qr = QuantumRegister(4)
cr = ClassicalRegister(4)

qcircuit = QuantumCircuit(qr, cr)

get_ipython().run_line_magic('matplotlib', 'inline')



#half adder circuit to add 1 and 1
qcircuit.x(qr[0])
qcircuit.x(qr[1])
#qcircuit.h(qr[2])
#qcircuit.cx(qr[0], qr[1])
qcircuit.ccx(qr[0], qr[1], qr[3]) #controllo, controllo, target
qcircuit.cx(qr[1], qr[2]) #controllo, target
qcircuit.cx(qr[0], qr[2])
qcircuit.measure(qr, cr)
qcircuit.draw(output = 'mpl')


#simulate it in local
simulator = Aer.get_backend('qasm_simulator')
result = execute(qcircuit, backend = simulator).result()
#from qiskit.tools.visualization import plot_histogram
plot_histogram(result.get_counts(qcircuit))



#load the IBM account to launch the circuit on a quantum computer
IBMQ.load_account() #load the account in order to execute the code on the
quantum computer


#select the backend
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_burlington')
job = execute(qcircuit, backend = qcomp, shots = 8192)
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

```
#here we can see how the circuit is transpiled on the quantum computer
from qiskit.visualization import plot_circuit_layout

qc_transpiled = transpile(qcircuit, backend = qcomp,
optimization_level=3)
plot_circuit_layout(qc_transpiled, qcomp, view = 'virtual')
```

```
#and the error map, thus we can choose if continue with this backend or
change it
from qiskit.visualization import plot_error_map
plot_error_map(qcomp) #we plot the error map of the backend selected
```

```
#to see the results

qresult = job.result()
plot_histogram(qresult.get_counts(qcircuit))
```

```
#interactive plot

from qiskit.visualization import iplot_histogram
iplot_histogram(qresult.get_counts(qcircuit))
```

```
#FULL ADDER
from qiskit import *
from qiskit import IBMQ
from qiskit.visualization import plot_histogram, plot_bloch_vector
get_ipython().run_line_magic('config', "InlineBackend.figure_format =
'svg' # Makes the images look nice")

qr = QuantumRegister(4)
cr = ClassicalRegister(4)

qcircuit = QuantumCircuit(qr, cr)

get_ipython().run_line_magic('matplotlib', 'inline')

#our circuit to implement the sum of 1, 0 and a carry in equal to 1


qcircuit.x(qr[0]) #set to 1 the 1st addend
qcircuit.x(qr[2]) #set to 1 the carry in
qcircuit.ccx(qr[0], qr[1], qr[3]) #control, control, target (Toffoli
gate)
qcircuit.cx(qr[0], qr[1]) #control, target (CNOT gate)
qcircuit.ccx(qr[1], qr[2], qr[3])
qcircuit.cx(qr[1], qr[2])
qcircuit.cx(qr[0], qr[1])
qcircuit.measure(qr, cr)
qcircuit.draw(output = 'mpl')


#in local
simulator = Aer.get_backend('qasm_simulator')
result = execute(qcircuit, backend = simulator).result()
print(result.get_counts(qcircuit))
plot_histogram(result.get_counts(qcircuit))




#load the account in order to launch our circuit on a real quantum
computer
IBMQ.load_account() #load the account in order to execute the code on the
quantum computer




#get the provider and use a job monitor
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne') #had to change from
imbq_london for more accuracy
```

```
job = execute(qcircuit, backend = qcomp, shots = 8192)
from qiskit.tools.monitor import job_monitor
job_monitor(job)


#see the results
qresult = job.result()
print(qresult.get_counts(qcircuit))
plot_histogram(qresult.get_counts(qcircuit))

#ADDER
from qiskit import *
from qiskit import IBMQ
from qiskit.visualization import plot_histogram, plot_bloch_vector
get_ipython().run_line_magic('config', "InlineBackend.figure_format =
'svg' # Makes the images look nice")

qr = QuantumRegister(7)
cr = ClassicalRegister(7)

qcircuit = QuantumCircuit(qr, cr)

get_ipython().run_line_magic('matplotlib', 'inline')


#this 4 statements are applied in order to see the superposition of all
qubits
#uncomment them to see the results
#qcircuit.h(qr[4])
#qcircuit.h(qr[5])
#qcircuit.h(qr[0])
#qcircuit.h(qr[1])

#sum of 10 and 10 in our quantum adder
qcircuit.x(qr[4])
qcircuit.x(qr[5])
qcircuit.ccx(qr[0], qr[1], qr[3]) #control, control, target
qcircuit.cx(qr[1], qr[2]) #control, target
qcircuit.cx(qr[0], qr[2])
qcircuit.ccx(qr[4], qr[5], qr[6])
qcircuit.cx(qr[4], qr[5])
qcircuit.ccx(qr[3], qr[5], qr[6])
qcircuit.cx(qr[5], qr[3])
qcircuit.cx(qr[4], qr[5])
qcircuit.swap(qr[3], qr[5])
qcircuit.swap(qr[2], qr[4])
qcircuit.swap(qr[1], qr[2])

#ENG
#with swaps I change my output in order to have a clear ouput
#starting from the MSQB (most significant Qbit)
#cout, somma full adder, somma half adder, b1, b0, a1, a0
#where a and b are the addends, the output is the match of carry out, sum
of FA and sum of HA

#IT
#con gli swap modifico la visione, avro' (partendo dal MSQb):
#cout, somma full adder, somma half adder, b1, b0, a1, a0
#dove gli a e b sono gli addendi e l'insieme di cout, somma FA e somma HA
sono il risultato finale
```

```
qcircuit.measure(qr, cr)
qcircuit.draw(output = 'mpl')


#launch it locally
simulator = Aer.get_backend('qasm_simulator')
result = execute(qcircuit, backend = simulator).result()

plot_histogram(result.get_counts(qcircuit))


#load the IBM account
IBMQ.load_account() #load the account in order to execute the code on the
quantum computer


#choose the quantum computer where you will launch your circuit
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne') #had to change from
imbq_london for more accuracy
job = execute(qcircuit, backend = qcomp, shots = 8192)
from qiskit.tools.monitor import job_monitor
job_monitor(job)


#to see how it is transpiled
from qiskit.visualization import plot_circuit_layout

qc_transpiled = transpile(qcircuit, backend = qcomp,
optimization_level=3)
plot_circuit_layout(qc_transpiled, qcomp, view = 'virtual')




#plot the error map and see if you have choose a quantum computer that
will satisfy your circuit
from qiskit.visualization import plot_error_map
plot_error_map(qcomp) #we plot the error map of the backend selected


#plot of the probabilities
qresult = job.result()
plot_histogram(qresult.get_counts(qcircuit))


#interactive plot
from qiskit.visualization import iplot_histogram
iplot_histogram(qresult.get_counts(qcircuit))
```