```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


print("Quantum Random Number Generator 1.0.7\nMade by Frederick Westinghouse\nWith API Token
Input 1.0\n\nInitializing...\n")
import pip
def import_or_install(qiskit):
    try:
        __import__(qiskit)
    except ImportError:
        pip.main(['install', qiskit])
import_or_install('qiskit')
def import_or_install(ipywidgets):
    try:
        __import__(ipywidgets)
    except ImportError:
        pip.main(['install', ipywidgets])
#     __ipywidgets_version__
import_or_install('ipywidgets')




# Importing standard Qiskit libraries and configuring account
import qiskit
from qiskit import*
from qiskit.tools.monitor import job_monitor
import webbrowser
import tkinter as tk
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
qiskit.__qiskit_version__
from qiskit import IBMQ
import time
from IPython.display import clear_output

pig="HelloHumans"
x=len(pig)
if x !=11:
    print(x)
print ('Please enter your IBMQ API Token. If you have run this program before and have already
done so press s to skip\nIf you are switching your API Token press d')

if pig=='d':
    IBMQ.delete_account()
while (len(pig) !=1 and len(pig) !=128):
    print('\nType "info" if you want a tutorial on getting an API Token.')
    pig=input()
    print(pig)
```

```
   if len(pig) <=3:
      break
   elif pig=='info':
      clear_output(wait=False)
      print ('An API token allows you to run your job on a real quantum computer at IBM. It links
this program to your IBMQ account.\n\n\nA webpage will open when you press any key.\n\nSign into
your IBMQ account or create one.\nAfter sign in, you will be taken to your account page.\nLook
for the blue button that says "Copy token".\nClick on that button. Now your API token has been
copied to your clipboard.\nCome back here after that and right click and select paste in the
input box that will have showed up.\nYour API Token will be pasted. Press enter and you are
done.\n')
      input('Press any key to open that website.')
      print('Opening web page in a new tab...\n')
      time.sleep(1)
      print("\nGo to the tab that is opening, and come back after you have pressed the blue 'Copy
token' button\n")
      time.sleep(3)
      import webbrowser
      new=2
      url='https://quantum-computing.ibm.com/account'
      webbrowser.open(url,new=new)
      time.sleep(1)
      clear_output(wait=False)
      time.sleep(6)
      clear_output(wait=False)
      continue
   elif (len(pig) != 128):
      clear_output(wait=False)
      print ("It seems like you are trying to enter an API Token, but "+pig+" is not a valid
token. Look again for your API token. It should be a 128 charecter gibberish string. Try again")
      time.sleep(4)
      clear_output(wait=False)
      continue
   else:

      text_file = open("IBMQ API Token.txt", "w")
      text_file.truncate()
      n = text_file.write(pig)
      text_file.close()

      infile=open('IBMQ API Token.txt','r')
      lines=infile.readlines()
      infile.close()
      list_q = lines
      str(list_q).strip('[]')
      ', '.join(map(str, list_q))
      lines_fin=('\n'.join(map(str, list_q)))
      print(lines_fin)


      IBMQ.save_account(lines_fin)
      if lines_fin != pig:
         print('Error. Try again.')
         continue
```

```
            break
        break

time.sleep(4)
clear_output(wait=False)
print ("Imports done.")
sim=Aer.get_backend("qasm_simulator")
IBMQ.load_account() #Loads the IBM account. Allows the circuit to be sent to a real quantum computer.
provider=IBMQ.get_provider('ibm-q' )
qcomp=provider.get_backend('ibmq_athens')
sizeofqandc=5
max_size=255
startofrange=1
endofrange=10
print ("Preparations Done!\n\n")
time.sleep(2)
clear_output(wait=False)



def Quantum_Random_Number_Generator():
    backend=sim #Enter 'sim' to run on a local simulator or 'qcomp' to run on a real IBM quantum computer. Overridden by next block.
    while True:
        try:
            print('Now you can choose the range of values that you want your random numbers to be in.\nIt can be any range, but the smallest value cannot be less than 0 and the largest cannot be more than 250.\nTwo boxes will show up. Enter one number at a time, and make sure that number is an integer!')
            start_range=int(input("Each random integer should have a value between\n"))
        except ValueError:
            print("\nSorry, I didn't understand that. Enter an integer please.\n")
            #better try again... Return to the start of the loop
            continue
        if start_range<0:
            print('\nError. Please enter a number greater than or equal to 0.\n')
            continue
        if start_range>250:
            print('\nError. Please enter a number less than or equal to 250.\n')
            continue
            #start_range was successfully parsed!
            #we're ready to exit the loop.
        else:
            break

    while True:
        try:
            end_range=int(input('and\n'))
        except ValueError:
            print("\nSorry, I didn't understand that. Enter an integer please.\n")
            #better try again... Return to the start of the loop
            continue
```

```
      if end_range<0:
         print('\nError. Please enter a number greater than or equal to 0.\n')
         continue
      if end_range>250:

         print('\nError. Please enter a number less than or equal to 250.\n')
         continue
      if end_range<start_range:
         print('\nError. Please enter a number that is greater than the beginning of the range.')
         continue
         #start_range was successfully parsed!
         #we're ready to exit the loop.
      if end_range==start_range:
         print ('\nError. Please enter a range of at least one. It defeats the purpose to have the
start and end of the range be the same.')
         time.sleep(2)
      else:
         break
   clear_output(wait=True)

   start_range_print=str(start_range)
   end_range_print=str(end_range)
   print('\nYou have chosen '+ start_range_print + '-' + end_range_print + ' as your range.')
   time.sleep(3)

   clear_output(wait=False)

   while True:
      try:
         number_of_R_numbers=int(input('\nPlease enter any number of random numbers that you
want to generate.\nExamples: Enter "10" if you want to generate around 10 random numbers, "17"
for around 17 numbers and so on.\nAny number above 0 works. The maximum is 200, but if you get an
error try again with a smaller number.\n'))
      except ValueError:
         clear_output(wait=False)
         print("\nSorry, I didn't understand that. Enter an integer please.\n")
         #better try again... Return to the start of the loop
         time.sleep(4)
         clear_output(wait=False)
         continue
      if number_of_R_numbers<0:
         clear_output(wait=False)
         print('\nError. Please enter a number greater than or equal to 0.\n')
         time.sleep(4)
         clear_output(wait=False)
         continue
      if number_of_R_numbers>200:
         clear_output(wait=False)
         print('\nError. Please enter a number less than or equal to 200.\n')
         time.sleep(4)
         clear_output(wait=False)
         continue
      else:
         break
```

```
    time.sleep(1.5)
    clear_output(wait=False)
    print('\nThe true number of random numbers may vary up to 14 percent from the value that you
put in.\n')
    time.sleep(2)

    #Finds the digit_factor based on number of single/double/triple digit numbers.
    #To find digit_factor find the difference between the total number of digits in the random numbers
and the number of whole numbers.

    if (start_range<10):
        fd1=10-start_range
        if end_range<=100:
            print('Y1')
            fd2=end_range-9
        if end_range>=100:
            print('Y2')
            fd2=100-9
    if end_range>=100:
        if end_range<1000: #Formality or just in case end_range maximum ever gets bigger.
            print('Y3')
            fd3=end_range-99
    if start_range<10:
            if end_range<100:
                print('Y4')
                df1=fd1+fd2
                ef1=fd2*2
                bf1=ef1+fd1
                cf1=bf1/df1
                digit_factor=cf1
    if start_range<10:
        if end_range>=100:
            if end_range<1000: #Formality
                print('Y5')
                rt1=fd1+fd2+fd3
                rg2=fd2*2
                rg3=fd3*3
                vf1=rg3+rg2+fd1
                dlp1=vf1/rt1
                digit_factor=dlp1
    if start_range>=10:
        if end_range>=100:
            if end_range<1000: #Formality
                print('Y6')
                ltt1=fd2+fd3
                lt2=fd2*2
                lt3=fd3*3
                vt1=lt2+lt3
                dpi1=vt1/ltt1
                digit_factor=dpi1
    if (start_range<10):
        if (end_range<10):
            print('N1')
```

```
        digit_factor=1
    if (start_range>=10):
        if (end_range<100):
            print('N2')
            digit_factor=2
    if (start_range>=100):
        if (end_range<1000): #This is a formality only.
            print('N3')
            digit_factor=3

    digit_factor_str=str(digit_factor)
    print('Digit Factor= '+digit_factor_str)

    difference=(end_range+1)-start_range
    percentage_of_whole=(difference*100)/256
    numberofnumbers_factor=100/percentage_of_whole
    numofnum_factor=numberofnumbers_factor*digit_factor
    number_of_Rnumbers_float=float(number_of_R_numbers)
    shot_num=numofnum_factor*number_of_Rnumbers_float
    shot_num=int(shot_num)
    base=8
    shot_num = base*round(shot_num/base)
    shot_num_str=str(shot_num)
    print("Going to run " + shot_num_str + " shots." )
    time.sleep(3)
    clear_output(wait=False)

    real_or_fake = ''
    while (real_or_fake != 'real' and real_or_fake != 'Real' and real_or_fake !="r" and
real_or_fake !="fake" and real_or_fake !="Fake" and real_or_fake !="f"):
        real_or_fake=input("Please enter 'real' to run on a real quantum computer for true
randomness or 'fake' to run on a local simulator.\n")
        if (real_or_fake != 'real' and real_or_fake != 'Real' and real_or_fake !="r" and
real_or_fake !="fake" and real_or_fake !="Fake" and real_or_fake !="f"):
            print ("Please enter 'real' or 'fake'. Not anything else!")
            time.sleep(3)
            clear_output(wait=False)
    if real_or_fake in ('fake','Fake','f'):
        backend=sim
    if real_or_fake=='fake':
        time.sleep(1.5)
        clear_output(wait=False)
    if real_or_fake=='Fake':
        time.sleep(1.5)
    if real_or_fake=='f':
        time.sleep(1.5)
        clear_output(wait=False)
    if real_or_fake=='real':
        time.sleep(1.5)
        clear_output(wait=False)
    if real_or_fake=='Real':
        time.sleep(1.5)
        clear_output(wait=False)
    if real_or_fake=='r':
```

```
        time.sleep(1.5)
        clear_output(wait=False)

    if real_or_fake in ('real','Real','r'):
        print("Now we can find the quantum computer with the smallest queue.\n")
        website=input('Press "y" to open the webpage that will show you the different quantum
computers and the length of their queues.\nNote: The quantum computers are called backends. The
backend called "qasm_simulator" at the end of the list\nis the local simulator, not a real
quantum computer at IBM\n"ibmq_armonk" does not have enough qubits. Do not choose it!\nPress n to
skip and load default quantum computer(ibmq_athens).\nPress s to skip to quantum computer
selection.\n')

        time.sleep(0.5)
        clear_output(wait=False)
        if (website=='n'):
            print('The number generator will run on the default quantum computer (ibmq_athens).\n')
            backend=qcomp
            time.sleep(3)
            clear_output(wait=False)
        if (website=='y'):
              print ('Opening web page in a new tab...\n')
              time.sleep(1)
              print("\nGo to the tab that is opening, and come back when you have found the quantum
computer with the shortest queue.\n")
              time.sleep(3)
              import webbrowser
              new=2
              url='https://quantum-computing.ibm.com/'
              webbrowser.open(url,new=new)
              time.sleep(1)
              clear_output(wait=False)
              time.sleep(6)
              clear_output(wait=False)

        quantum_computer=''
        while (quantum_computer != "ibmq_santiago"
            and quantum_computer != "santiago"
            and quantum_computer != "santiago"
            and quantum_computer != "ibmq_athens"
            and quantum_computer != "athens"
            and quantum_computer != "vigo"
            and quantum_computer != "ibmq_vigo"
            and quantum_computer != "ibmq_valencia"
            and quantum_computer != "16_melbourne"
            and quantum_computer != "melbourne"
            and quantum_computer != "ibmq_burlington"
        #   and quantum_computer != "burlington"
        #   and quantum_computer != "ibmq_essex"
        #   and quantum_computer != "essex"
            and quantum_computer != "ourense"
            and quantum_computer != "ibmq_ourense"
            and quantum_computer != "ibmq_5_yorktown_ibmqx2"
            and quantum_computer != "ibmq_5_yorktown"
            and quantum_computer != "5_yorktown"
```

```
        and quantum_computer != "yorktown"
        and quantum_computer != "ibmq_16_melbourne"):
        quantum_computer=input("Enter the exact name of the quantum computer (backend) with
the smallest queue.\nIf you are unsure leave this blank, hit enter, and the default (ibmq_athens)
will be used.\n")
        if (quantum_computer=="ibmq_santiago"):
           qcomp=provider.get_backend('ibmq_santiago')
           break
        elif (quantum_computer=="ibmq_athens"):
           qcomp=provider.get_backend('ibmq_athens')
           break

        elif (quantum_computer=="ibmq_vigo"):
           qcomp=provider.get_backend('ibmq_vigo')

        elif (quantum_computer=="ibmq_valencia"):
           qcomp=provider.get_backend('ibmq_valencia')

        elif (quantum_computer=="ibmq_16_melbourne"):
           qcomp=provider.get_backend('ibmq_16_melbourne')
           print("\nYou have chosen the most powerful quantum computer that is available to the
public!")
           time.sleep(3)
        elif (quantum_computer=="ibmq_melbourne"):
           qcomp=provider.get_backend('ibmq_16_melbourne')
           print("\nYou have chosen the most powerful quantum computer that is available to the
public!")
           time.sleep(3)
        elif (quantum_computer=="ibmq_burlington"):
           qcomp=provider.get_backend('ibmq_burlington')

        elif (quantum_computer=="ibmq_essex"):
           qcomp=provider.get_backend('ibmq_essex')

        elif (quantum_computer=="ibmq_ourense"):
           qcomp=provider.get_backend('ibmq_ourense')

        elif (quantum_computer=="ibmq_5_yorktown-ibmqx2"):
           qcomp=provider.get_backend('ibmq_5_yorktown')

        elif (quantum_computer=="ibmq_armonk"):
           print('This quantum computer only has 1 qubit. We need 5. Pick another quantum
computer.\n')
           time.sleep(3)
        elif (quantum_computer=="ibmq_qasm_simulator"):
           print ('You have chosen the simulator on your local computer.\n Pick another quantum
computer.\n')
           time.sleep(3)
        elif (quantum_computer=="santiago"):
           qcomp=provider.get_backend('ibmq_santiago')

        elif (quantum_computer=="athens"):
           qcomp=provider.get_backend('ibmq_athens')
```

```
        elif (quantum_computer=="vigo"):
            qcomp=provider.get_backend('ibmq_vigo')

        elif (quantum_computer=="valencia"):
            qcomp=provider.get_backend('ibmq_valencia')

        elif (quantum_computer=="melbourne"):
            qcomp=provider.get_backend('ibmq_16_melbourne')
            print("\nYou have chosen the most powerful quantum computer that is available to the
public!")
            time.sleep(3)
        elif (quantum_computer=="16_melbourne"):
            qcomp=provider.get_backend('ibmq_16_melbourne')
            print("\nYou have chosen the most powerful quantum computer that is available to the
public!")
            time.sleep(3)
        elif (quantum_computer=="burlington"):
            qcomp=provider.get_backend('ibmq_burlington')

        elif (quantum_computer=="essex"):
            qcomp=provider.get_backend('ibmq_essex')

        elif (quantum_computer=="ourense"):
            qcomp=provider.get_backend('ibmq_ourense')

        elif (quantum_computer=="ibmq_5_yorktown"):
            qcomp=provider.get_backend('ibmq_5_yorktown')

        elif (quantum_computer=="5_yorktown"):
            qcomp=provider.get_backend('ibmq_5_yorktown')

        elif (quantum_computer=="yorktown"):
            qcomp=provider.get_backend('ibmq_5_yorktown')

        elif (quantum_computer=="armonk"):
            print('This quantum computer only has 1 qubit. We need 5. Pick another quantum
computer.\n')
            time.sleep(3)
        elif (quantum_computer=="qasm_simulator"):
            print ('You have chosen the simulator on your local computer.\n Pick another quantum
computer.\n')
        elif (quantum_computer=="simulator"):
            print ('You have chosen the simulator on your local computer.\n Pick another quantum
computer.\n')
            time.sleep(3)
        else:
            clear_output(wait=False)
            print('Error. There is no quantum computer at IBM named ' +quantum_computer+'.
Please try again.')
            time.sleep(4)
            clear_output(wait=False)

    if (real_or_fake in ('real','Real')):
        #qcomp_selector()
```

```
        backend=qcomp

    time.sleep(1)
    clear_output(wait=False)


    q = QuantumRegister(sizeofqandc)
    c = ClassicalRegister(sizeofqandc)
    circuit = QuantumCircuit(q,c)
    circuit.h(q) # Applies hadamard gate to all qubits
    circuit.measure(q,c) # Measures all qubits
    print ("Please wait. Randomness takes time.")
    time.sleep(1.5)
    clear_output(wait=False)
    print("Executing Job...\nPlease wait a few minutes. Exact time will depend on the busyness of
the quantum computer\n")

    if (real_or_fake in ('real','Real')):
        if (backend==qcomp):
            print ("Run 1: Real Quantum Computer ("+quantum_computer+")!\n")
    if (backend==sim):
        print("Run 1: Simulator of Quantum Computer (qasm_simulator).\n\n")

    # Now run the circuit

    job= execute(circuit, backend=backend, shots=shot_num, memory=True)
    job_monitor(job)

    rawvalues_calculation = job.result().get_memory()

    # now take all results from qubit 1 as bytes in the first 1k, qubit 2 in the 2nd k, and so on
    if (max_size==255):
        binarybytes_ibm = [];
        for rollypolly in range(0, 5):
            for i in range(0, len(rawvalues_calculation), 8):
                b = int(rawvalues_calculation[i][rollypolly] + rawvalues_calculation[i+1][rollypolly] +
rawvalues_calculation[i+2][rollypolly] + rawvalues_calculation[i+3][rollypolly] +
rawvalues_calculation[i+4][rollypolly] + rawvalues_calculation[i+5][rollypolly] +
rawvalues_calculation[i+6][rollypolly] + rawvalues_calculation[i+7][rollypolly], 2)
                binarybytes_ibm.append(b)

    print('\n')
    clear_output(wait=False)
    print('\nThe Job Has Finished Executing.\nRandom Numbers:')


    list_of_ints = binarybytes_ibm
    str(list_of_ints).strip('[]')
    ', '.join(map(str, list_of_ints))
    random_number_string=('\n'.join(map(str, list_of_ints)))

    text_file = open("Random.txt", "w")
    n = text_file.write(random_number_string)
    text_file.close()
```

```
    #limit the range of numbers that are shown.
    infile=open('Random.txt','r')
    lines=infile.readlines()
    infile.close()
    for line in lines:
        number=int(line)
        if start_range-1<number<end_range+1:
            print(number)


    print ("\n\nDone! You may run again or copy the numbers and close the tab.")

def import_or_install(tkinter):
    try:
        __import__(tkinter)
    except ImportError:
        pip.main(['install', tkinter])
        __tkinter_version__
#import_or_install('tkinter')

#root= tk.Tk()

#canvas1 = tk.Canvas(root, width = 300, height = 300)
#canvas1.pack()

#def start ():
#    label1 = tk.L
#    abel(root, text= 'Hello World!', fg='green', font=('helvetica', 12, 'bold'))
#    canvas1.create_window(720, 1280, window=label1)

#button1 = tk.Button(text='Click to Start Quantum Random Number
Generator',command=Quantum_Random_Number_Generator, bg='brown',fg='white')
#canvas1.create_window(150, 150, window=button1)

#root.destroy
#root.mainloop()

#clear_output(wait=False)


Quantum_Random_Number_Generator()


# In[ ]:




# In[ ]:
```