```
#-----------------------------------------------------------------------------------------------------------
# Import necessary modules
#-----------------------------------------------------------------------------------------------------------
import sys
from qiskit import QuantumProgram
import Qconfig
import Basic_gates
import math
from random import randint
import control_unitaries
import xlsxwriter
#-----------------------------------------------------------------------------------------------------------
# global variables
#-----------------------------------------------------------------------------------------------------------
Counts = 0
A = 0
Ran_Quantum_period_finding = 0
global m
m = 0
#-----------------------------------------------------------------------------------------------------------
# The function to compute GCD using Euclid's method
# Input : Two number to X and Y for which a GCD is to be computed
# Output : GCD of two given numbers
#-----------------------------------------------------------------------------------------------------------
def gcd(x,y):
    while y != 0:
```

```
        (x, y) = (y, x % y)
    return x


#------------------------------------------------------------------------------------------------------------------
# The function to construct the unitary based on a
# Input : Quantum program object, the Circuit name and the quantum register name, and a
# Output : None. The relevant circuit is made.
#------------------------------------------------------------------------------------------------------------------
def cmod(Quantum_program_object,Circuit_name,Quantum_register_name,a):
# Get the circuit and the quantum register by name
 qc = Quantum_program_object.get_circuit(Circuit_name)
 qr = Quantum_program_object.get_quantum_register(Quantum_register_name)


# Construct unitary based on a
 if a == 2:
  qc.cswap(qr[4],qr[3],qr[2])
  qc.cswap(qr[4],qr[2],qr[1])
  qc.cswap(qr[4],qr[1],qr[0])
 if a == 4 or a == 11 or a == 14:
  qc.cswap(qr[4],qr[2],qr[0])
  qc.cswap(qr[4],qr[3],qr[1])
  qc.cx(qr[4],qr[3])
  qc.cx(qr[4],qr[2])
  qc.cx(qr[4],qr[1])
  qc.cx(qr[4],qr[0])
 if a == 7:
```

```
  qc.cswap(qr[4],qr[1],qr[0])
  qc.cswap(qr[4],qr[2],qr[1])
  qc.cswap(qr[4],qr[3],qr[2])
  qc.cx(qr[4],qr[3])
  qc.cx(qr[4],qr[2])
  qc.cx(qr[4],qr[1])
  qc.cx(qr[4],qr[0])
 if a == 8:
  qc.cswap(qr[4],qr[1],qr[0])
  qc.cswap(qr[4],qr[2],qr[1])
  qc.cswap(qr[4],qr[3],qr[2])
 if a == 13:
  qc.cswap(qr[4],qr[3],qr[2])
  qc.cswap(qr[4],qr[2],qr[1])
  qc.cswap(qr[4],qr[1],qr[0])
  qc.cx(qr[4],qr[3])
  qc.cx(qr[4],qr[2])
  qc.cx(qr[4],qr[1])
  qc.cx(qr[4],qr[0])


#------------------------------------------------------------------------------------------------------
# The function to compute QFT
# Input : Circuit, quantum bits, and number of quantum bits
# Output : None. Circuit is created and saved
#------------------------------------------------------------------------------------------------------
def
```

```
qft(Quantum_program_object,Circuit_name,Quantum_register_name,Smallest_Quantum_register_n
umber,Size_of_QFT):
 # Get the circuit and the quantum register by name
 qc = Quantum_program_object.get_circuit(Circuit_name)
 qr = Quantum_program_object.get_quantum_register(Quantum_register_name)
 s = Smallest_Quantum_register_number
 for j in range(Size_of_QFT):
  for k in range(j):
   qc.cu1(math.pi/float(2**(j-k)), qr[s+j], qr[s+k])
  qc.h(qr[s+j])

#----------------------------------------------------------------------------------------------------------
# The function to find period using the Quantum computer
# Input : a and N for which the period is to be computed.
# Output : period r of the function a^x mod N
#----------------------------------------------------------------------------------------------------------
def period(a,N):
 global Ran_Quantum_period_finding
 Ran_Quantum_period_finding = 1
# Create the first QuantumProgram object instance.
 qp = QuantumProgram()
 #qp.set_api(Qconfig.APItoken, Qconfig.config["url"])
# TO DO : generalize the number of qubits and give proper security against rogue input.
# Create the first Quantum Register called "qr" with 12 qubits
 qr = qp.create_quantum_register('qr', 5)
# Create your first Classical Register  called "cr" with 12 bits
```

```
 cr = qp.create_classical_register('cr', 3)


# Create the first Quantum Circuit called "qc" involving your Quantum Register "qr"
# and the Classical Register "cr"
 qc = qp.create_circuit('Period_Finding', [qr], [cr])


# Get the circuit and the registers by name
 Shor1 = qp.get_circuit('Period_Finding')
 Q_reg = qp.get_quantum_register('qr')
 C_reg = qp.get_classical_register('cr')


# Create the circuit for period finding
# Initialize qr[0] to |1>
 Shor1.x(Q_reg[0])


# Step one : apply a**4 mod 15
 Shor1.h(Q_reg[4])
# Controlled Identity on the remaining 4 qubits. Which is equivalent to doing nothing
 Shor1.h(Q_reg[4])
 Shor1.measure(Q_reg[4],C_reg[0])
# Reinitialize to |0>
 Shor1.reset(Q_reg[4])


# Step two : apply a**2 mod 15
 Shor1.h(Q_reg[4])
# Controlled unitary. Apply a mod 15 twice.
```

```
 for k in range(2):
  cmod(qp,'Period_Finding','qr',a)
 if C_reg[0] == 1 :
  Shor1.u1(pi/2.0,Q_reg[4])
 Shor1.h(Q_reg[4])
 Shor1.measure(Q_reg[4],C_reg[1])
# Reinitialize to |0>
 Shor1.reset(Q_reg[4])


# Step three : apply 11 mod 15
 Shor1.h(Q_reg[4])
# Controlled unitary. Apply a mod 15
 cmod(qp,'Period_Finding','qr',a)
# Feed forward and measure
 if C_reg[1] == 1 :
  Shor1.u1(pi/2.0,Q_reg[4])
 if C_reg[0] == 1 :
  Shor1.u1(pi/4.0,Q_reg[4])
 Shor1.h(Q_reg[4])
 Shor1.measure(Q_reg[4],C_reg[2])


 # Run the circuit
 #qp.set_api(Qconfig.APItoken, Qconfig.config['url']) # set the APIToken and API url
 simulate = qp.execute(["Period_Finding"], backend="local_qasm_simulator", shots=1,timeout=500)
 simulate.get_counts("Period_Finding")
 #print(simulate)
```

```
 data = simulate.get_counts("Period_Finding")
 #print(data)
 data = list(data.keys())
 #print(data)
 r = int(data[0])
 #print(r)
 l = gcd(2**3,r)
 #print(l)
 r = int((2**3)/l)
 #print(r)
 return r


#-----------------------------------------------------------------------------------------------------------
# The main function to compute factors
# Input : The number to be factored, N
# Output : Factors of the number
#-----------------------------------------------------------------------------------------------------------
def Factorize_N(N):
 factors = [0,0]
#-----------------------------------------------------------------------------------------------------------
# Step 1 : Determine the number of bits based on N; n = [log2(N)]
#-----------------------------------------------------------------------------------------------------------
 n = math.ceil(math.log(N,2))
#-----------------------------------------------------------------------------------------------------------
# Step 2 : Check if N is even. In that case return 2 and the remaining number as factors
#-----------------------------------------------------------------------------------------------------------
```

```
 if N % 2 == 0:
  factors = [2,N/2]
  return factors
#---------------------------------------------------------------------------------------------------------
# Step 3 : Check if N is of the form P^(k), where P is some prime factor. In that case return P and k.
#---------------------------------------------------------------------------------------------------------
# The step has been eliminated for simulation purposes.
#---------------------------------------------------------------------------------------------------------
# Step 4 : Choose a random number between 2...(N-1).
#---------------------------------------------------------------------------------------------------------
 while True:
  a = randint(2,N-1)
  global A
  A = a
#---------------------------------------------------------------------------------------------------------
# Step 5 : Take GCD of a and N. t = GCD(a,N)
#---------------------------------------------------------------------------------------------------------
  t = gcd(N,a)
  if t > 1:
   factors = [t,N/t]
   return factors
#---------------------------------------------------------------------------------------------------------
# Step 6 : t = 1. Hence, no common period. Find Period using Shor's method
#---------------------------------------------------------------------------------------------------------
  r = period(a,N)
  if (r%2 == 0) and (((a**(r/2))+1)%N != 0) and (r != 0) and (r != 8):
```

```
   break
  global Counts
  Counts = Counts + 1
 factor_1 = gcd((a**(r/2))+1,N)
 factor_2 = N/factor_1
 factors = [factor_1,factor_2]
 return factors


#-----------------------------------------------------------------------------------------------------------
# Running the Shor's algorithm version 1
#-----------------------------------------------------------------------------------------------------------


#-----------------------------------------------------------------------------------------------------------
# Step 0 : Take the input N
#-----------------------------------------------------------------------------------------------------------
factors_list = list()
A_used = list()
Ran_QPF = list()
Total_counts = list()
if __name__ == '__main__':
 #global m
 for m in range(100):
  N = 15
  factors_found = Factorize_N(N)
  factors_list.append(factors_found)
  #ws.write(row,col,A)
```

```
  #ws.write(row,col+1,Counts)

  #ws.write(row,col+2,factors[0])

  #ws.write(row,col+3,factors[1])

  #ws.write(row,col+4,Ran_Quantum_period_finding)

  #row = row + 1

  #print("The Number being factorized is 15")

  #print("Factors are = ",factors)

  #print("Number of times the quantum circuit did not give correct period = ",Counts)

  #print ("The parameter a used = ", A)

  print("Run ", m)

  A_used.append(A)

  Ran_QPF.append(Ran_Quantum_period_finding)

  Total_counts.append(Counts)

  Counts = 0

  Ran_Quantum_period_finding = 0



if m == 99:

 wb = xlsxwriter.Workbook('log.xlsx')

 ws = wb.add_worksheet('Data')

 row = 0

 col = 0

 ws.write(row,col,'a used for factorizing')

 ws.write(row,col+1,'Number of times the quantum circuit did not give correct period')

 ws.write(row,col+2,'Factor1')

 ws.write(row,col+3,'Factor2')
```

```
ws.write(row,col+4,'Ran_Quantum_period_finding?')
row = row + 1

for k in range(100):
 ws.write(row,col,A_used[k])
 ws.write(row,col+1,Total_counts[k])
 ws.write(row,col+2,factors_list[k][0])
 ws.write(row,col+3,factors_list[k][1])
 ws.write(row,col+4,Ran_QPF[k])
 row = row + 1

wb.close()
```