```
"""
=====================
Quantum Circuit Simulator
=====================
"""
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot, ion, show
from matplotlib.colors import ListedColormap
import random
from collections import Counter
import itertools
from numpy.linalg import multi_dot

"Ground state"
def ggs(num_qubits): #which is short for get_ground_state(num_qubits):
    "return vector of size 2**num_qubits with all zeroes except first element which is 1"
    ground_state = np.array([np.zeros(2**num_qubits)])
    ground_state[0][0] = 1
    return ground_state.T

"Quantum Logic Gates"

"One Qubit Gates"
X = np.array([[0, 1], [1, 0]])
Y = np.array([[0, -1j], [1j, 0]])
Z = np.array([[1, 0], [0, -1]])
H = np.dot(2**(-0.5), np.array([[1, 1],[1, -1]]))
I = np.array([[1, 0], [0, 1]])
def U(theta, nx, ny, nz):
    "theta is the rotation angle and (nx, ny, nz) should be a unit vector in Cartesian coordinates in the
Bloch sphere describing the rotation axis"
    return 1j*np.cos(theta/2)*I + np.sin(theta/2)*(nx*X + ny*Y + nz*Z)

"Product Gates"
def gpg(arg): #which is short for get_product_gate(*arg):
    #input the gates that you want to apply using square brackets and quotation marks (e.g.
["X","H","Z"] if you want X to the first qubit, H to the second and Z to the final qubit)
    "return matrix of size 2**num_qubits by 2**num_qubits"
    if len(arg) == 2:
        return np.kron(eval(arg[-2]), eval(arg[-1]))
    else:
        argWithoutFirst = arg[1:]
        return np.kron(eval(arg[0]), gpg(argWithoutFirst))

"Controlled Gates"
"Define useful matrices, such as the projection operations |0><0| and |1><1|"
P0X0 = np.array([[1, 0], [0, 0]])
P1X1 = np.array([[0, 0], [0, 1]])
```

```
def gcg(*arg): #which is short for get_controlled_gate(*arg):
    #input the gates that you want to apply using quotation marks (e.g. "c","X","I","I" if you want the first to be the control and X to be (conditionally) applied to the second qubit while the third and fourth qubit is uninvolved
    "return matrix of size 2**num_qubits by 2**num_qubits"
    n = 0
    for x in arg:
        n = n + x.count('c')
    if n == 0:
        print("If there are no controls, please use gpg to implement the circuit correctly.")
    if n > 1:
        print("I cannot apply gates with multiple controls.")
    else:
        controlled_gate = np.zeros((2**len(arg),2**len(arg)))
        arg0 = ["P0X0" if x == 'c' else 'I' for x in arg]
        arg1 = ["P1X1" if x == 'c' else x for x in arg]
        controlled_gate = gpg(arg0) + gpg(arg1)
        return controlled_gate

"Compile and run the programme"
def comp(circuit): #which is short for compile
    circuit_eval = [eval(circuit[x]) for x in range(len(circuit))]
    if len(circuit_eval) == 1:
        return circuit_eval[0]
    else:
        return np.linalg.multi_dot(circuit_eval)

def run_programme(circuit, ground_state):
    final_state = np.dot(comp(circuit), ground_state)
    return final_state

"Get the counts"

"This creates a list of all the possible outputs"
def bin_list(n):
     return ["".join(seq) for seq in itertools.product("01", repeat = int(n))]

def measure_all(state_vector, num_shots):
    weights = [np.abs(state_vector.tolist()[x])**2 for x in range(len(state_vector.tolist()))]
    n = np.log2(len(state_vector.tolist()))
    outputs = random.choices(bin_list(n), weights, k = num_shots)
    cwzm = Counter(outputs) #which is short for counts_without_zeros_mentioned
    list_of_all_counts = ['%s : %d' % (x, cwzm[x]) for x in bin_list(n)]
    return list_of_all_counts

"This is where the user defines what they want to simulate"

"Define the ground state for the desired number qubits"
my_num_qubits = 3
my_ground_state = ggs(my_num_qubits)

"Define a programme consisting of well-defined gates"
```

```
#Note that gpg requires square brackets but gcg does not
my_circuit = ["gpg([\"X\",\"I\",\"I\"])",
            "gcg(\"c\",\"I\",\"X\")",
            "gcg(\"c\",\"U(np.pi,1,0,0)\",\"I\")",
            "gpg([\"I\",\"I\",\"H\"])"
            ]

"Define the desired number of shots (i.e. the number of times the circuit is run)"
my_shots = 1000

"This tells Python to print the user's input, run their programme and print the results"
print(my_circuit, end = '\n \n')

print(measure_all(run_programme(my_circuit[::-1], my_ground_state), my_shots))
```