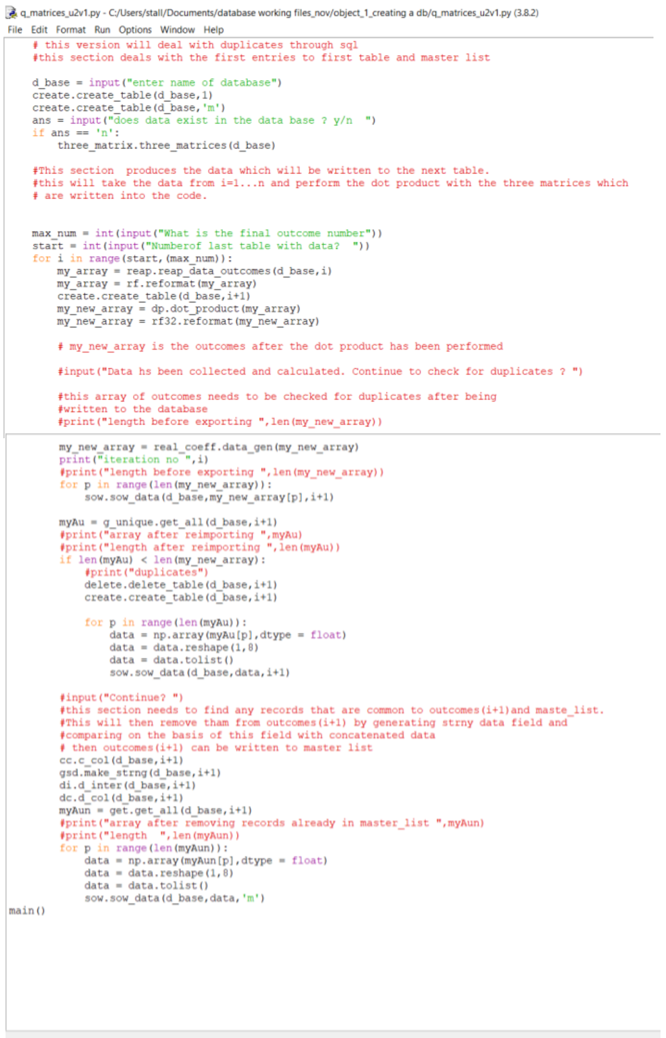
Exploring Quantum Gates   Project 1

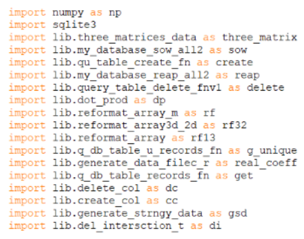
My Database

The code for this part of the project is in file: q_matrices_u2v1.py. It uses many of the modules outlined earlier. These modules were imported as shown below right.



q_matrices_u2v1.py - C:/Users/stall/Documents/database working files_nov/object_1_creating a db/q_matrices_u2v1.py (3.8.2)

```python
    # this version will deal with duplicates through sql
    #this section deals with the first entries to first table and master list

    d_base = input("enter name of database")
    create.create_table(d_base,1)
    create.create_table(d_base,'m')
    ans = input("does data exist in the data base ? y/n  ")
    if ans == 'n':
        three_matrix.three_matrices(d_base)

    #This section  produces the data which will be written to the next table.
    #this will take the data from i=1...n and perform the dot product with the three matrices which
    # are written into the code.


    max_num = int(input("What is the final outcome number"))
    start = int(input("Numberof last table with data?  "))
    for i in range(start,(max_num)):
        my_array = reap.reap_data_outcomes(d_base,i)
        my_array = rf.reformat(my_array)
        create.create_table(d_base,i+1)
        my_new_array = dp.dot_product(my_array)
        my_new_array = rf32.reformat(my_new_array)

        # my_new_array is the outcomes after the dot product has been performed

        #input("Data hs been collected and calculated. Continue to check for duplicates ? ")

        #this array of outcomes needs to be checked for duplicates after being
        #written to the database
        #print("length before exporting ",len(my_new_array))

        my_new_array = real_coeff.data_gen(my_new_array)
        print("iteration no ",i)
        #print("length before exporting ",len(my_new_array))
        for p in range(len(my_new_array)):
            sow.sow_data(d_base,my_new_array[p],i+1)

        myAu = g_unique.get_all(d_base,i+1)
        #print("array after reimporting ",myAu)
        #print("length after reimporting ",len(myAu))
        if len(myAu) < len(my_new_array):
            #print("duplicates")
            delete.delete_table(d_base,i+1)
            create.create_table(d_base,i+1)

            for p in range(len(myAu)):
                data = np.array(myAu[p],dtype = float)
                data = data.reshape(1,8)
                data = data.tolist()
                sow.sow_data(d_base,data,i+1)

        #input("Continue? ")
        #this section needs to find any records that are common to outcomes(i+1)and maste_list.
        #This will then remove tham from outcomes(i+1) by generating strny data field and
        #comparing on the basis of this field with concatenated data
        # then outcomes(i+1) can be written to master list
        cc.c_col(d_base,i+1)
        gsd.make_strng(d_base,i+1)
        di.d_inter(d_base,i+1)
        dc.d_col(d_base,i+1)
        myAun = get.get_all(d_base,i+1)
        #print("array after removing records already in master_list ",myAun)
        #print("length  ",len(myAun))
        for p in range(len(myAun)):
            data = np.array(myAun[p],dtype = float)
            data = data.reshape(1,8)
            data = data.tolist()
            sow.sow_data(d_base,data,'m')
main()
```

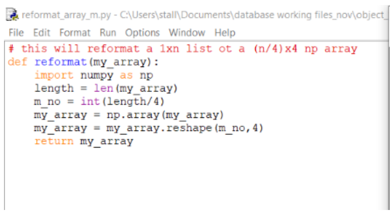


```python
import numpy as np
import sqlite3
import lib.three_matrices_data as three_matrix
import lib.my_database_sow_all2 as sow
import lib.qu_table_create_fn as create
import lib.my_database_reap_all2 as reap
import lib.query_table_delete_fnv1 as delete
import lib.dot_prod as dp
import lib.reformat_array_m as rf
import lib.reformat_array3d_2d as rf32
import lib.reformat_array as rf13
import lib.q_db_table_u_records_fn as g_unique
import lib.generate_data_filec_r as real_coeff
import lib.q_db_table_records_fn as get
import lib.delete_col as dc
import lib.create_col as cc
import lib.generate_strngy_data as gsd
import lib.del_intersction_t as di
```

step 1 create database or open database if it exists

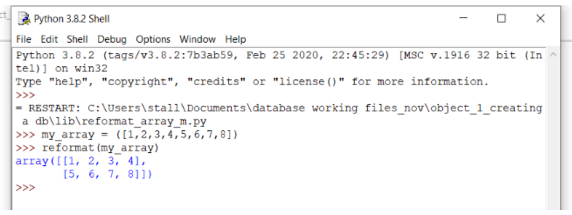
step 2 add h t and s matrices to first table and master table if no data is in the database

step 3 read the data from the last table and reformat



reformat_array_m.py - C:\Users\stall\Documents\database working files_nov\object_

```python
# this will reformat a 1xn list ot a (n/4)x4 np array
def reformat(my_array):
    import numpy as np
    length = len(my_array)
    m_no = int(length/4)
    my_array = np.array(my_array)
    my_array = my_array.reshape(m_no,4)
    return my_array
```

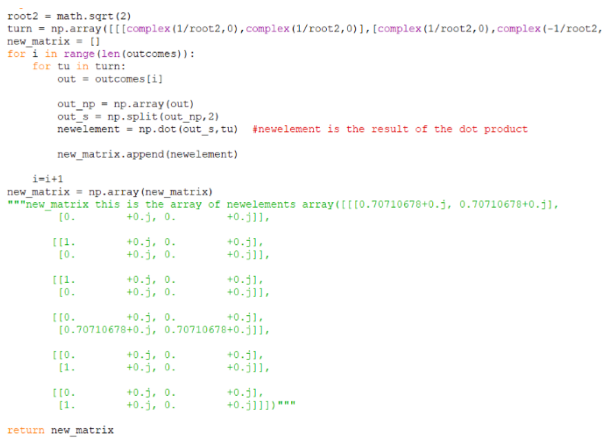


Python 3.8.2 Shell

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\stall\Documents\database working files_nov\object_1_creating
 a db\lib\reformat_array_m.py
>>> my_array = ([1,2,3,4,5,6,7,8])
>>> reformat(my_array)
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>>
```

step 4 create a new table for the results which will be generated in the next few steps step 5 run the dot product module.

This module has a function *dot_product(outcomes)* where outcomes needs to be a list of lists[[1,0,0,0],[0,0,1,0]]. Each elemental list has 4 complex numbers 2 representing the contents of each row of a 2x2 matrix. This format is consistent with the output format of my_database_reap_all2 . The 1x4 shape is then reshaped to 2x2 matrix each time an element is used for the dot product

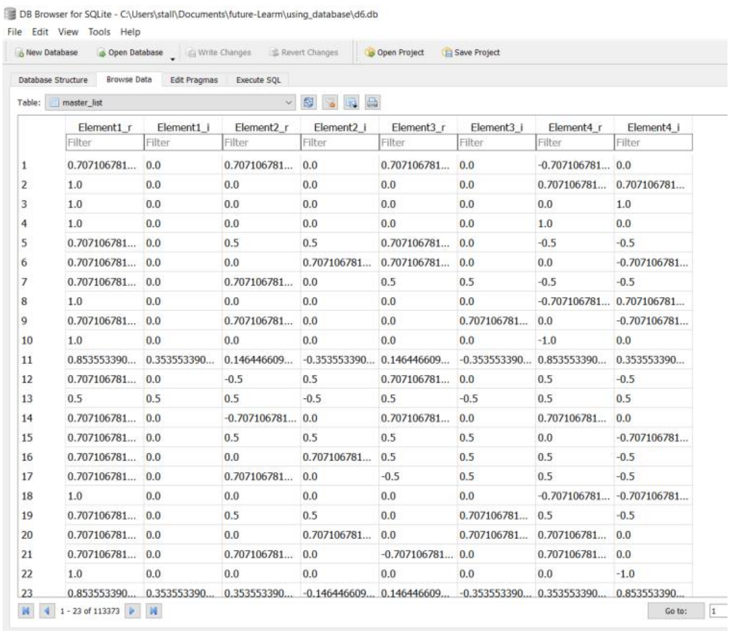


```
root2 = math.sqrt(2)
turn = np.array([[[complex(1/root2,0),complex(1/root2,0)],[complex(1/root2,0),complex(-1/root2,
new_matrix = []
for i in range(len(outcomes)):
    for tu in turn:
        out = outcomes[i]

        out_np = np.array(out)
        out_s = np.split(out_np,2)
        newelement = np.dot(out_s,tu)  #newelement is the result of the dot product

        new_matrix.append(newelement)

    i=i+1
new_matrix = np.array(new_matrix)
"""new_matrix this is the array of newelements array([[[0.70710678+0.j, 0.70710678+0.j],
        [0.        +0.j, 0.        +0.j]],

       [[1.        +0.j, 0.        +0.j],
        [0.        +0.j, 0.        +0.j]],

       [[1.        +0.j, 0.        +0.j],
        [0.        +0.j, 0.        +0.j]],

       [[0.        +0.j, 0.        +0.j],
        [0.70710678+0.j, 0.70710678+0.j]],

       [[0.        +0.j, 0.        +0.j],
        [1.        +0.j, 0.        +0.j]],

       [[0.        +0.j, 0.        +0.j],
        [1.        +0.j, 0.        +0.j]]])"""

return new_matrix
```

step 6 In order to write this array of complex number to the database, it had to be reshaped and coefficients stored as a data list. This was done using modules reformat_array3d_2d and reformat_array

step 7 Using sql queries to remove duplicates both from the new table an also any duplication between this table and the master list

step 8 running sql queries to write unique data to the database.



	Element1_r	Element1_i	Element2_r	Element2_i	Element3_r	Element3_i	Element4_r	Element4_i
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	0.707106781...	0.0	0.707106781...	0.0	0.707106781...	0.0	-0.707106781...	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.707106781...	0.707106781...
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
5	0.707106781...	0.0	0.5	0.5	0.707106781...	0.0	-0.5	-0.5
6	0.707106781...	0.0	0.0	0.707106781...	0.707106781...	0.0	0.0	-0.707106781...
7	0.707106781...	0.0	0.707106781...	0.0	0.5	0.5	-0.5	-0.5
8	1.0	0.0	0.0	0.0	0.0	0.0	-0.707106781...	0.707106781...
9	0.707106781...	0.0	0.707106781...	0.0	0.0	0.707106781...	0.0	-0.707106781...
10	1.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0
11	0.853553390...	0.353553390...	0.146446609...	-0.353553390...	0.146446609...	-0.353553390...	0.853553390...	0.353553390...
12	0.707106781...	0.0	-0.5	0.5	0.707106781...	0.0	0.5	-0.5
13	0.5	0.5	0.5	-0.5	0.5	-0.5	0.5	0.5
14	0.707106781...	0.0	-0.707106781...	0.0	0.707106781...	0.0	0.707106781...	0.0
15	0.707106781...	0.0	0.5	0.5	0.5	0.5	0.0	-0.707106781...
16	0.707106781...	0.0	0.0	0.707106781...	0.5	0.5	0.5	-0.5
17	0.707106781...	0.0	0.707106781...	0.0	-0.5	0.5	0.5	-0.5
18	1.0	0.0	0.0	0.0	0.0	0.0	-0.707106781...	-0.707106781...
19	0.707106781...	0.0	0.5	0.5	0.0	0.707106781...	0.5	-0.5
20	0.707106781...	0.0	0.0	0.707106781...	0.0	0.707106781...	0.707106781...	0.0
21	0.707106781...	0.0	0.707106781...	0.0	-0.707106781...	0.0	0.707106781...	0.0
22	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0
23	0.853553390...	0.353553390...	0.353553390...	-0.146446609...	0.146446609...	-0.353553390...	0.353553390...	0.853553390...

1 - 23 of 113373