Please use this Google doc to code during your interview. To free your hands for coding, we recommend that you use a headset or a phone with speaker option.

def reverse\_it(values):

 n = len(values)

 for i in range(n):

   values[i] = values[n - i]

   values[n - i] = values[i]

 return values

if \_\_name\_\_ == '\_\_main\_\_':

 values = [1, 25, 12, 80]

 rev = reverse\_it(values)

 print('Reverse: {} -> {}'.format(values, rev))

Debuged:

def reverse\_it(values):

 values\_copy = values[:]

 n = len(values\_copy )

 for i in range(n//2):

   values\_copy [i], values\_copy [n - i - 1] = values\_copy [n - i - 1], values\_copy [i]

 return values\_copy

if \_\_name\_\_ == '\_\_main\_\_':

 values = [1, 25, 12, 80]

 rev = reverse\_it(values)

 print('Reverse: {} -> {}'.format(values, rev))

eg. 1

values = [1, 23, 12, 80]

values[0] = values[4 - 0 - 1] ⇒ [80,25,12,1]

values[1] = values[4 - 1 - 1] => [80,12,25,1]

eg. 2

values = [1,23,12,80,40]

eg. 3

values = []

2

======

     40

 12       80

1   23

     35

Please use this Google doc to code during your interview. To free your hands for coding, we recommend that you use a headset or a phone with speaker option.

17

1 - 1

2 - 4

3 - 9

4 - 16

5 - 25

bounds = [4 - 5]

4.5 \* 4.5 =

4.25 \* 4.25 =

4.125 \* 4.125 =

sqrt(4, 0.2): ans = 1.8 - 2.2

def square\_root(num, pres):

upper\_bound = 0

lower\_bound = num

result = (upper\_bound - lower\_bound)/2

while True:

if (result - pres)\*\*2 < num  and num < (result + pres)\*\*2 :

break

if num > result\*\*2:

lower\_bound = result

result = (upper\_bound - lower\_bound)/2

elif num < result\*\*2:

upper\_lower = result

result = (upper\_bound - lower\_bound)/2

else:

return result

return result

O ~ log(num)

Please use this Google doc to code during your interview. To free your hands for coding, we recommend that you use a headset or a phone with speaker option.

Input: User

Output: User with the most friends in common with input user

friend\_data = {

‘user1’ : [‘friend1’, ‘friend2’],

‘friend1’ : [‘user1’,‘friend3’,’friend2’, ’friend4’]

‘friend2’ : [‘user1’,‘friend3’,’friend2’, ’friend5’]

}

def find\_mutual\_friends(user,friend\_data):

count = {}

if user not in friend\_data:

return None

my\_friends = friends\_data[user] + user

for friend in friends\_data[user]:

if friend in friend\_data:

uncommon\_friends = set(friends\_data[friend]) - set(my\_friends)

for uncom in uncommon\_friend:

if uncom in count:

count[uncom] += 1

else:

count[uncom] = 1

if count:

mutual\_frinds = sorted(count, key=count.\_\_getitem\_\_, reverse = True)

return mutual\_friend[0]

return None

eg.

user = ‘user1’

…

count = {}

my\_friends = [‘friend1’, ‘friend2’, ‘user1’]

uncommon\_frind = [‘friend3’, ‘friend4’]

count = {

‘friend3’ : 1,

‘frind4’ : 1

}

uncommon\_frind = [‘friend3’, ‘friend5’]

count = {

‘friend3’ : 2,

‘frind4’ : 1,

‘friend5’ : 1

}

mutual\_friend = [‘friend3’,’friend4’,’friend5’]

friend\_data = {

‘user1’ : [‘friend1’, ‘friend2’],

‘friend1’ : [‘user1’,’friend2’]

‘friend2’ : [‘user1’,’friend1’]

}

#####################

friend\_data = {

‘user1’ : [‘friend1’, ‘friend2’],

‘friend1’ : [‘user1’,‘friend3’,’friend2’, ’friend4’]

‘friend2’ : [‘user1’,‘friend3’,’friend2’, ’friend5’]

}

personal\_hobby = {

‘user1’ : [‘swimming’, ‘football’],

‘friend5’ : [playing\_guitar, singing],

….

}

def find\_mutual\_friends(user,friend\_data, personal\_hobby):

count = {}

if user not in friend\_data:

return None

my\_friends = friends\_data[user] + user

for friend in friends\_data[user]:

if friend in friend\_data:

uncommon\_friends = set(friends\_data[friend]) - set(my\_friends)

for uncom in uncommon\_friend:

if uncom in count:

count[uncom] += 1

else:

count[uncom] = 1

if uncom in personal\_hobby:

# get all the common bobbies by intersection of two sets

common\_hobbies = set(personal\_hobby[user]) | set(personal\_hobby[uncom])

count[uncom] += len(common\_hobbies)

if count:

mutual\_frinds = sorted(count, key=count.\_\_getitem\_\_, reverse = True)

return mutual\_friend[0]

return None