Adrian Lievano submission for OPENAI Coding Challenge

This notebook breakdowns the process for extracting all of the usernames in a given database.

In [490]:
```
query = lambda prefix: [d for d in database if d.startswith(prefix)][:5]
```

CODING CHALLENGE PROBLEM DESCRIPTION

You're on a website (such as Github!) with a text field which autocompletes usernames as you type. Under the hood, there's an API call which takes in the prefix of a username and then returns all usernames which start with that prefix, lexicographically sorted and truncated at 5 results.

Your task is to use this API call to dump the entire user database, specifically:

Implement the extract function in autocomplete.py. extract should return the whole user database, making calls to query under the hood.

Notes:

You can assume all valid usernames are comprised of lowercase ASCII letters (a-z). Assume that queries to the API are expensive and should be minimized, but it's more important to have a correct answer and well-structured solution than an optimal answer. You are welcome to include any tests or documentation that you'd normally provide when checking in to a shared codebase. Submit your answer as a secret Gist. Please do not make your answers public (and please do not look for solutions from others).

In [ ]:
```
def extract(query):
    """Implement this method using the `query` API call, for example:
    query("abracadar") #=> ["abracadara"] using the default query method in main()
    """
    # YOUR CODE HERE

    return [...]

def main():
    """Runs your solution -- no need to update (except to maybe change the database)."""
    # Simple implementation of the autocomplete API
    database = ["abracadara", "al", "alice", "alicia", "allen", "alter", "altercation", "bob", "eve",
 "evening", "event", "eventually", "mallory"]
    query = lambda prefix: [d for d in database if d.startswith(prefix)][:5]
    assert extract(query) == database

main()
```

Feature Extraction Code

The Code below extracts 1st, 2nd, and 3rd orders features (character combinations) that lead to identifying every username in the database

Rationale Behind the Algorithm

This algorithm takes in letters, or caps, A-Z, a-z, numbers, special characters -- because usernames can be anything, right? Then, it proceeds to identify lower-order features to higher-order ones, systematically decreasing the database by removing usernames from the original database that are below the number of characters defined by a given filter. For example, in the string, 'Alter', 'A', 'Al', 'Alt', are 1st, 2nd, and 3rd order features, respectively.

After extracting a feature, for example, 'A', it uses the discovered nth-features to query the dataset on the remaining_database from the (nth-1) feature.

We're calling quieries on the set of identified 1st, 2nd, 3rd, ... nth order features (characters in a word) that is unique to our database. By adding checker flags between each feature-order, we can minimize the number of characters needed to identify all the users in the database, thus minimizing the number of times we have to call query API.

If there wasn't a minimum, or a maximum, on the number of characters, I think we can continue to draw out the nth order of terms to selectively identify more complex usernames and extract more complex names and sized databases.

In [496]:
```
#Adrian Lievano's CODE STARTS HERE
#Set & Sorts database, in case it doesn't contain unique usernames or if it's not properly ordered
database = ['ASD','ASD','GOHASD','GOHAN','ASDD,','GTGTG','GTGsdsd','lo','adrian','guano','jimcarrey',
'grunt','MasterChief','AAAAAAAA','AAA','AAB','AABC','AAACC','AAACD','AAACVX','$','a','b','c','d',"abr
acadara", "al", "alice","#312Yomama","#412Yomama","#512Yomama","#612Yomama","#712Yomama","#812Yomama"
, "alicia", "allen", "alter",'913KILLA','123Banyo','xx#Chaosxx#','23Yawwhey' "altercation","amap","am
map","azap","azzopa","AcePilota19993","bob", "bzaop", "eve", "evening", "event", "eventually", "mallo
ry"]
database = sorted(set(database))

def getWords(data):
    words=[]
    for i in range(len(data)):
        word=data[i]
        words.append(word)
    return words

   ## for strings in range(len(word.split())):
      #  #print(strings)

def getLetters(allWords):

    #Initialize chars and Letters matrix
    chars = []
    fls=[]

    for x in range(len(allWords)):
        listChar=list(allWords[x])
        chars.append(listChar)
    #return chars

    for i in range(len(allWords)):
        fl=chars[i][0:]
        fls.append(fl)
    return fls

#def check_ifs (fls):
```

In [464]:
```
remainder_database=(database)
words=getWords(database)
letters=getLetters(words)

##########################################def getFirstOrderLetters:#################################
########
#Initialize list for 1st order combination terms
flc=[]
print("  ")
print('Remaining usernames: {} '.format(remainder_database))

for i in range(len(letters[:][0:])):
    flc.append(letters[i][0])
#Gets unique 1-letter combinations & sorts them is lexicographically sorted
flc = sorted(list(set(flc)))
print("  ")
print( 'This is the set of the first order terms: {}'.format(flc))

#Calls query on each 1st order combination in order and removes from the original database
for i in range(len(flc)):
    remainder_database=sorted(list(set(remainder_database)-set(query(flc[i]))))
print("  ")
print('Remaining usernames: {} '.format(remainder_database))
print("    ")

#PRINT SUCCESS MESAGE IF ALL USERS HAVE BEEN IDENTIFIED
if (len(remainder_database)<1):
    print('Congratulations! We identified all users in our database')
else:
    print('Find higher order features')

##########################################def getSecondOrderLetters:#################################
#########

#Creates 1st char removal list
    remove_flc = []
    for i in range(len(database)):
        if (len(letters[i][0:]) < 2):
            remove_flc.append(database[i])

#filters out 1-char characters
    remainder_database=sorted(list(set(database)-set(remove_flc)))
```

```
    print("  ")
    print("We wil filter out {} terms before discovering 2nd order features".format(remove_flc))
    print("  ")
    print('This is the post-filtered database: {}'.format(remainder_database))
    print("  ")

#Initialize list for 2nd order combination terms
    words=getWords(remainder_database)
    letters=getLetters(words)
    slc=[]

    for i in range(len(letters[:][0:])):
        slc.append(letters[i][0]+letters[i][1])
    #Gets unique 2-letter combinations & sorts them is lexicographically sorted
    slc = sorted(list(set(slc)))
    print("    ")
    print( 'This is the set of the second order terms: {}'.format(slc))


#Calls query on each 2nd order combination in order and removes from the original database
    for i in range(len(slc)):
        remainder_database=sorted(list(set(remainder_database)-set(query(slc[i]))))
    print("  ")
    print('Remaining usernames: {} '.format(remainder_database))
    print("  ")

#PRINT SUCCESS MESSAGE IF ALL USERS HAVE BEEN IDENTIFIED
#############Check for remaining usernames; proceed to filter using third order combinations#########
####
    if (len(remainder_database)<1):
        print('Congratulations! We identified all users in our database only using 1st and 2nd order
 terms')
    else:
        print('Find higher order terms')
##########################################def getThirdOrderLetters:#################################
########
        print(" ")
        print(letters)
        #This restarts the database and allows the 3rd order feature extraction algorithm to remove 1
-char and 2-char length usernames
        remainder_database=(database)
        words=getWords(database)
        letters=getLetters(words)
```

```
        print(" ")

        #Creates 2nd char removal list
        remove_slc = []
        for i in range(len(database)):
            if (len(letters[i][0:]) <3 and len(letters[i][0:]) > 1 ):
                remove_slc.append(database[i])

        #filters out 1-char characters
        remainder_database=sorted(list(set(database)-set(remove_slc)-set(remove_flc)))
        print("  ")
        print("We wil filter out {} and {} terms before discovering 3rd order features".format(remove
_flc,remove_slc))
        print("  ")
        print('This is the post-filtered database: {}'.format(remainder_database))
        print("  ")
        print("  ")

        #Reinitialize list for 3rd order feature extraction
        words=getWords(remainder_database)
        letters=getLetters(words)
        tlc=[]

        for j in range(len(letters)):
            tlc.append(letters[j][0]+letters[j][1]+letters[j][2])
        #Gets unique 3-letter combinations & sorts them is lexicographically sorted
        tlc = sorted(list(set(tlc)))
        print("  ")
        print( 'This is the set of the third order terms: {}'.format(tlc))

        #Calls query on each 3rd order combination in order and removes from the original database
        for i in range(len(tlc)):
            remainder_database=sorted(list(set(remainder_database)-set(query(tlc[i]))))
        print("  ")
        print('Remaining usernames: {} '.format(remainder_database))
        print("    ")


        #PRINT SUCCESS MESSAGE IF ALL USERS HAVE BEEN IDENTIFIED
        if (len(remainder_database)<1):
            print('Congratulations! We identified all users in our database using 1st, 2nd, and 3rd o
rder features')
        else:
```

```
            print('Find higher order terms')

qu = len(flc)+len(slc)+len(tlc)
#########OUTPUT ALL USERS IN LEX ORDER
print("    ")
print('Our features: ')
print("    ")
print(flc)
print("    ")
print(slc)
print("    ")
print(tlc)
print("    ")
print("    ")
print('Number of querys to extract features: {}'.format(qu))
print('Number of usernames: {}'.format(len(database)))
```

```
Remaining usernames: ['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yomama', '#812Yom
ama', '$', '123Banyo', '23Yawwheyaltercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AAACD', 'AAA
CVX', 'AAB', 'AABC', 'ASD', 'ASDD,', 'AcePilota19993', 'GOHAN', 'GOHASD', 'GTGTG', 'GTGsdsd', 'Maste
rChief', 'a', 'abracadara', 'adrian', 'al', 'alice', 'alicia', 'allen', 'alter', 'amap', 'ammap', 'a
zap', 'azzopa', 'b', 'bob', 'bzaop', 'c', 'd', 'eve', 'evening', 'event', 'eventually', 'grunt', 'gu
ano', 'jimcarrey', 'lo', 'mallory', 'xx#Chaosxx#']

This is the set of the first order terms: ['#', '$', '1', '2', '9', 'A', 'G', 'M', 'a', 'b', 'c',
'd', 'e', 'g', 'j', 'l', 'm', 'x']

Remaining usernames: ['#812Yomama', 'AAB', 'AABC', 'ASD', 'ASDD,', 'AcePilota19993', 'alicia', 'alle
n', 'alter', 'amap', 'ammap', 'azap', 'azzopa']

Find higher order features

We wil filter out ['$', 'a', 'b', 'c', 'd'] terms before discovering 2nd order features

This is the post-filtered database: ['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yo
mama', '#812Yomama', '123Banyo', '23Yawwheyaltercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AA
ACD', 'AAACVX', 'AAB', 'AABC', 'ASD', 'ASDD,', 'AcePilota19993', 'GOHAN', 'GOHASD', 'GTGTG', 'GTGsds
d', 'MasterChief', 'abracadara', 'adrian', 'al', 'alice', 'alicia', 'allen', 'alter', 'amap', 'amma
p', 'azap', 'azzopa', 'bob', 'bzaop', 'eve', 'evening', 'event', 'eventually', 'grunt', 'guano', 'ji
mcarrey', 'lo', 'mallory', 'xx#Chaosxx#']


This is the set of the second order terms: ['#3', '#4', '#5', '#6', '#7', '#8', '12', '23', '91', 'A
A', 'AS', 'Ac', 'GO', 'GT', 'Ma', 'ab', 'ad', 'al', 'am', 'az', 'bo', 'bz', 'ev', 'gr', 'gu', 'ji',
'lo', 'ma', 'xx']

Remaining usernames: ['AAB', 'AABC']

Find higher order terms

[['#', '3', '1', '2', 'Y', 'o', 'm', 'a', 'm', 'a'], ['#', '4', '1', '2', 'Y', 'o', 'm', 'a', 'm',
'a'], ['#', '5', '1', '2', 'Y', 'o', 'm', 'a', 'm', 'a'], ['#', '6', '1', '2', 'Y', 'o', 'm', 'a',
'm', 'a'], ['#', '7', '1', '2', 'Y', 'o', 'm', 'a', 'm', 'a'], ['#', '8', '1', '2', 'Y', 'o', 'm',
'a', 'm', 'a'], ['1', '2', '3', 'B', 'a', 'n', 'y', 'o'], ['2', '3', 'Y', 'a', 'w', 'w', 'h', 'e',
'y', 'a', 'l', 't', 'e', 'r', 'c', 'a', 't', 'i', 'o', 'n'], ['9', '1', '3', 'K', 'I', 'L', 'L',
'A'], ['A', 'A', 'A'], ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A'], ['A', 'A', 'A', 'C', 'C'], ['A',
'A', 'A', 'C', 'D'], ['A', 'A', 'A', 'C', 'V', 'X'], ['A', 'A', 'B'], ['A', 'A', 'B', 'C'], ['A',
'S', 'D'], ['A', 'S', 'D', 'D', ','], ['A', 'c', 'e', 'P', 'i', 'l', 'o', 't', 'a', '1', '9', '9',
'9', '3'], ['G', 'O', 'H', 'A', 'N'], ['G', 'O', 'H', 'A', 'S', 'D'], ['G', 'T', 'G', 'T', 'G'],
```

```
['G', 'T', 'G', 's', 'd', 's', 'd'], ['M', 'a', 's', 't', 'e', 'r', 'C', 'h', 'i', 'e', 'f'], ['a',
'b', 'r', 'a', 'c', 'a', 'd', 'a', 'r', 'a'], ['a', 'd', 'r', 'i', 'a', 'n'], ['a', 'l'], ['a', 'l',
'i', 'c', 'e'], ['a', 'l', 'i', 'c', 'i', 'a'], ['a', 'l', 'l', 'e', 'n'], ['a', 'l', 't', 'e',
'r'], ['a', 'm', 'a', 'p'], ['a', 'm', 'm', 'a', 'p'], ['a', 'z', 'a', 'p'], ['a', 'z', 'z', 'o',
'p', 'a'], ['b', 'o', 'b'], ['b', 'z', 'a', 'o', 'p'], ['e', 'v', 'e'], ['e', 'v', 'e', 'n', 'i',
'n', 'g'], ['e', 'v', 'e', 'n', 't'], ['e', 'v', 'e', 'n', 't', 'u', 'a', 'l', 'l', 'y'], ['g', 'r',
'u', 'n', 't'], ['g', 'u', 'a', 'n', 'o'], ['j', 'i', 'm', 'c', 'a', 'r', 'r', 'e', 'y'], ['l',
'o'], ['m', 'a', 'l', 'l', 'o', 'r', 'y'], ['x', 'x', '#', 'C', 'h', 'a', 'o', 's', 'x', 'x', '#']]


We wil filter out ['$', 'a', 'b', 'c', 'd'] and ['al', 'lo'] terms before discovering 3rd order feat
ures

This is the post-filtered database: ['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yo
mama', '#812Yomama', '123Banyo', '23Yawwheyaltercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AA
ACD', 'AAACVX', 'AAB', 'AABC', 'ASD', 'ASDD,', 'AcePilota19993', 'GOHAN', 'GOHASD', 'GTGTG', 'GTGsds
d', 'MasterChief', 'abracadara', 'adrian', 'alice', 'alicia', 'allen', 'alter', 'amap', 'ammap', 'az
ap', 'azzopa', 'bob', 'bzaop', 'eve', 'evening', 'event', 'eventually', 'grunt', 'guano', 'jimcarre
y', 'mallory', 'xx#Chaosxx#']



This is the set of the third order terms: ['#31', '#41', '#51', '#61', '#71', '#81', '123', '23Y',
'913', 'AAA', 'AAB', 'ASD', 'Ace', 'GOH', 'GTG', 'Mas', 'abr', 'adr', 'ali', 'all', 'alt', 'ama', 'a
mm', 'aza', 'azz', 'bob', 'bza', 'eve', 'gru', 'gua', 'jim', 'mal', 'xx#']

Remaining usernames: []

Congratulations! We identified all users in our database using 1st, 2nd, and 3rd order features

Our features:

['#', '$', '1', '2', '9', 'A', 'G', 'M', 'a', 'b', 'c', 'd', 'e', 'g', 'j', 'l', 'm', 'x']

['#3', '#4', '#5', '#6', '#7', '#8', '12', '23', '91', 'AA', 'AS', 'Ac', 'GO', 'GT', 'Ma', 'ab', 'a
d', 'al', 'am', 'az', 'bo', 'bz', 'ev', 'gr', 'gu', 'ji', 'lo', 'ma', 'xx']

['#31', '#41', '#51', '#61', '#71', '#81', '123', '23Y', '913', 'AAA', 'AAB', 'ASD', 'Ace', 'GOH',
'GTG', 'Mas', 'abr', 'adr', 'ali', 'all', 'alt', 'ama', 'amm', 'aza', 'azz', 'bob', 'bza', 'eve', 'g
ru', 'gua', 'jim', 'mal', 'xx#']
```

```
Number of querys to extract features: 80
Number of usernames: 52
```

Rationale Behind the Algorithm

This algorithm takes in letters, or caps, A-Z, a-z, numbers, special characters -- because usernames can be anything, right? Then, it proceeds to identify lower-order features to higher-order ones, systematically decreasing the database by filtering usernames from the original database that are below the number of characters defined by a given filter. After, it uses the discovered nth-features to query the dataset on the remaining_database from the (nth-1) feature.

We're calling quieries on the set of identified 1st, 2nd, 3rd, ... nth order features (characters in a word) that is unique to our database. By adding checker flags between each feature-order, we can minimize the number of characters needed to identify all the users in the database, thus minimizing the number of times we have to call query API.

If there wasn't a minimum, or a maximum, on the number of characters, I think we can continue to draw out the nth order of terms to selectively identify more complex usernames and extract more complex names and sized databases.

GETTING FINAL OUTPUTS</H1>

The code below reconstructs the database by using nth features obtained from previous feature extraction code.

In [492]:
```
#OUTPUT ALL USERS IN LEX ORDER
print("    ")
print('Our features')
print(flc)
print("    ")
print(slc)
print("    ")
print(tlc)

output = []

#Logic: Call query on each ith element of first order features; check if len(remaining usernames) =
 0;
#if not, continue to loop until first order features run out; if there still exists usernames, call q
ueries on
#each jth element of second order features; check if len(remaining_database)=0;
#if not, continue to loop until second order featuers run out;

print("    ")
print("    ")

#Rebuilding portion of our database using 1st order features
for i in range(len(flc)):
    output=sorted(output+query(flc[i]))
print(" ")
print('This is our dataset after calling the query API with our 1st order features:')
print(" ")
print(output)

#Checks if output is less than the entire database; finds unique additions using 2nd order terms and
 rebuilds output
if (len(output)<len(database)):
    for j in range(len(slc)):
        output=sorted(set(output+query(slc[j])))

print(" ")
print('This is our dataset after calling the query API with our 2nd order features:')
print(" ")
print(output)


#Checks if output is less than the entire database; finds unique additions using 3rd order terms and
```

```
 rebuilds output
if (len(output)<len(database)):
    for k in range(len(tlc)):
        output=sorted(set(output+query(tlc[k])))

print(" ")
print('This is our dataset after calling the query API with our 3rd order features:')
print(" ")
print(output)

print(" ")
print('This is our original dataset:')
print(" ")
print(database)

assert output == sorted(database)
```

```
Our features
['#', '$', '1', '2', '9', 'A', 'G', 'M', 'a', 'b', 'c', 'd', 'e', 'g', 'j', 'l', 'm', 'x']

['#3', '#4', '#5', '#6', '#7', '#8', '12', '23', '91', 'AA', 'AS', 'Ac', 'GO', 'GT', 'Ma', 'ab', 'a
d', 'al', 'am', 'az', 'bo', 'bz', 'ev', 'gr', 'gu', 'ji', 'lo', 'ma', 'xx']

['#31', '#41', '#51', '#61', '#71', '#81', '123', '23Y', '913', 'AAA', 'AAB', 'ASD', 'Ace', 'GOH',
'GTG', 'Mas', 'abr', 'adr', 'ali', 'all', 'alt', 'ama', 'amm', 'aza', 'azz', 'bob', 'bza', 'eve', 'g
ru', 'gua', 'jim', 'mal', 'xx#']



This is our dataset after calling the query API with our 1st order features:

['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yomama', '$', '123Banyo', '23Yawwheyal
tercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AAACD', 'AAACVX', 'GOHAN', 'GOHASD', 'GTGTG',
'GTGsdsd', 'MasterChief', 'a', 'abracadara', 'adrian', 'al', 'alice', 'b', 'bob', 'bzaop', 'c', 'd',
'eve', 'evening', 'event', 'eventually', 'grunt', 'guano', 'jimcarrey', 'lo', 'mallory', 'xx#Chaosxx
#']

This is our dataset after calling the query API with our 2nd order features:

['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yomama', '#812Yomama', '$', '123Bany
o', '23Yawwheyaltercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AAACD', 'AAACVX', 'ASD', 'ASD
D,', 'AcePilota19993', 'GOHAN', 'GOHASD', 'GTGTG', 'GTGsdsd', 'MasterChief', 'a', 'abracadara', 'adr
ian', 'al', 'alice', 'alicia', 'allen', 'alter', 'amap', 'ammap', 'azap', 'azzopa', 'b', 'bob', 'bza
op', 'c', 'd', 'eve', 'evening', 'event', 'eventually', 'grunt', 'guano', 'jimcarrey', 'lo', 'mallor
y', 'xx#Chaosxx#']

This is our dataset after calling the query API with our 3rd order features:

['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yomama', '#812Yomama', '$', '123Bany
o', '23Yawwheyaltercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AAACD', 'AAACVX', 'AAB', 'AAB
C', 'ASD', 'ASDD,', 'AcePilota19993', 'GOHAN', 'GOHASD', 'GTGTG', 'GTGsdsd', 'MasterChief', 'a', 'ab
racadara', 'adrian', 'al', 'alice', 'alicia', 'allen', 'alter', 'amap', 'ammap', 'azap', 'azzopa',
'b', 'bob', 'bzaop', 'c', 'd', 'eve', 'evening', 'event', 'eventually', 'grunt', 'guano', 'jimcarre
y', 'lo', 'mallory', 'xx#Chaosxx#']

This is our original dataset:

['#312Yomama', '#412Yomama', '#512Yomama', '#612Yomama', '#712Yomama', '#812Yomama', '$', '123Bany
o', '23Yawwheyaltercation', '913KILLA', 'AAA', 'AAAAAAAA', 'AAACC', 'AAACD', 'AAACVX', 'AAB', 'AAB
```

```
C', 'ASD', 'ASDD,', 'AcePilota19993', 'GOHAN', 'GOHASD', 'GTGTG', 'GTGsdsd', 'MasterChief', 'a', 'ab
racadara', 'adrian', 'al', 'alice', 'alicia', 'allen', 'alter', 'amap', 'ammap', 'azap', 'azzopa',
'b', 'bob', 'bzaop', 'c', 'd', 'eve', 'evening', 'event', 'eventually', 'grunt', 'guano', 'jimcarre
y', 'lo', 'mallory', 'xx#Chaosxx#']
```

Submission of Extract Algorithm </h1>

The code below integrates the above functions and logic into the extract function.

In [495]:
```
def extract(query):
    """Implement this method using the `query` API call, for example:
    query("abracadar") #=> ["abracadara"] using the default query method in main()
    """
    # YOUR CODE HERE
    database = ["abracadara", "al", "alice", "alicia", "allen", "alter", "altercation", "bob", "eve",
 "evening", "event", "eventually", "mallory"]
    database = sorted(set(database))

    def getWords(data):
        words=[]
        for i in range(len(data)):
            word=data[i]
            words.append(word)
        return words

    def getLetters(allWords):
        #Initialize chars and Letters matrix
        chars = []
        fls=[]
        for x in range(len(allWords)):
            listChar=list(allWords[x])
            chars.append(listChar)
        #return chars
        for i in range(len(allWords)):
            fl=chars[i][0:]
            fls.append(fl)
        return fls


    remainder_database=(database)
    words=getWords(database)
    letters=getLetters(words)

##########################################def getFirstOrderLetters:#################################
########
    #Initialize list for 1st order combination terms
    flc=[]
    print("  ")
    print('Remaining usernames: {} '.format(remainder_database))

    for i in range(len(letters[:][0:])):
```

```
        flc.append(letters[i][0])
    #Gets unique 1-letter combinations & sorts them is lexicographically sorted
    flc = sorted(list(set(flc)))
    print("  ")
    print( 'This is the set of the first order terms: {}'.format(flc))

    #Calls query on each 1st order combination in order and removes from the original database
    for i in range(len(flc)):
        remainder_database=sorted(list(set(remainder_database)-set(query(flc[i]))))
    print("  ")
    print('Remaining usernames: {} '.format(remainder_database))
    print("    ")

    #PRINT SUCCESS MESAGE IF ALL USERS HAVE BEEN IDENTIFIED
    if (len(remainder_database)<1):
        print('Congratulations! We identified all users in our database')
    else:
        print('Find higher order features')
##########################################def getSecondOrderLetters:#################################
#########
        #Creates 1st char removal list
        remove_flc = []
        for i in range(len(database)):
            if (len(letters[i][0:]) < 2):
                remove_flc.append(database[i])

    #filters out 1-char characters
        remainder_database=sorted(list(set(database)-set(remove_flc)))
        print("  ")
        print("We will filter out {} terms before discovering 2nd order features".format(remove_flc))
        print("  ")
        print('This is the post-filtered database: {}'.format(remainder_database))
        print("  ")

    #Initialize list for 2nd order combination terms
        words=getWords(remainder_database)
        letters=getLetters(words)
        slc=[]

        for i in range(len(letters[:][0:])):
            slc.append(letters[i][0]+letters[i][1])
        #Gets unique 2-letter combinations & sorts them is lexicographically sorted
        slc = sorted(list(set(slc)))
```

```
        print("    ")
        print( 'This is the set of the second order terms: {}'.format(slc))


    #Calls query on each 2nd order combination in order and removes from the original database
        for i in range(len(slc)):
            remainder_database=sorted(list(set(remainder_database)-set(query(slc[i]))))
        print("  ")
        print('Remaining usernames: {} '.format(remainder_database))
        print("  ")

    #PRINT SUCCESS MESSAGE IF ALL USERS HAVE BEEN IDENTIFIED
    #############Check for remaining usernames; proceed to filter using third order combinations#####
########
        if (len(remainder_database)<1):
            print('Congratulations! We identified all users in our database only using 1st and 2nd or
der terms')
        else:
            print('Find higher order terms')
    ##########################################def getThirdOrderLetters:############################
############

            print(" ")
            print(letters)
            #This restarts the database and allows the 3rd order feature extraction algorithm to remo
ve 1-char and 2-char length usernames
            remainder_database=(database)
            words=getWords(database)
            letters=getLetters(words)
            print(" ")

            #Creates 2nd char removal list
            remove_slc = []
            for i in range(len(database)):
                if (len(letters[i][0:]) <3 and len(letters[i][0:]) > 1 ):
                    remove_slc.append(database[i])

            #filters out 1-char characters
            remainder_database=sorted(list(set(database)-set(remove_slc)-set(remove_flc)))
            print("  ")
            print("We will filter out {} and {} terms before discovering 3rd order features".format(r
emove_flc,remove_slc))
            print("  ")
```

```
            print('This is the post-filtered database: {}'.format(remainder_database))
            print("  ")

            #Reinitialize list for 3rd order feature extraction
            words=getWords(remainder_database)
            letters=getLetters(words)
            tlc=[]

            for j in range(len(letters)):
                tlc.append(letters[j][0]+letters[j][1]+letters[j][2])
            #Gets unique 3-letter combinations & sorts them is lexicographically sorted
            tlc = sorted(list(set(tlc)))
            print("  ")
            print( 'This is the set of the third order terms: {}'.format(tlc))

            #Calls query on each 3rd order combination in order and removes from the original databas
e
            for i in range(len(tlc)):
                remainder_database=sorted(list(set(remainder_database)-set(query(tlc[i]))))
            print("  ")
            print('Remaining usernames: {} '.format(remainder_database))
            print("    ")


            #PRINT SUCCESS MESSAGE IF ALL USERS HAVE BEEN IDENTIFIED
            if (len(remainder_database)<1):
                print('Congratulations! We identified all users in our database using 1st, 2nd, and 3
rd order features')
            else:
                print('Find higher order terms')

    qu = len(flc)+len(slc)+len(tlc)
    #########OUTPUT ALL USERS IN LEX ORDER

    print("    ")
    print('Our features: ')
    print("    ")
    print(flc)
    print("    ")
    print(slc)
    print("    ")
    print(tlc)
    print("    ")
```

```
    print('Number of querys to extract features: {}'.format(qu))
    print('Number of usernames: {}'.format(len(database)))

    #OUTPUT ALL USERS IN LEX ORDER
    print("    ")
    print('Our features:')
    print(flc)
    print("    ")
    print(slc)
    print("    ")
    print(tlc)

    output = []

    #Logic: Call query on each ith element of first order features; check if len(remaining usernames)
 = 0;
    #if not, continue to loop until first order features run out; if there still exists usernames, ca
ll queries on
    #each jth element of second order features; check if len(remaining_database)=0;
    #if not, continue to loop until second order featuers run out;

    print("    ")

    #Rebuilding portion of our database using 1st order features
    for i in range(len(flc)):
        output=sorted(output+query(flc[i]))
    print(" ")
    print('This is our dataset after using our 1st order features to rebuild the original dataset:')
    print(" ")
    print(output)

    #Checks if output is less than the entire database; finds unique additions using 2nd order terms
 and rebuilds output
    if (len(output)<len(database)):
        for j in range(len(slc)):
            output=sorted(set(output+query(slc[j])))

    print(" ")
    print('This is our dataset after using our 2nd order features to rebuild the original dataset:')
    print(" ")
    print(output)
```

```
    #Checks if output is less than the entire database; finds unique additions using 3rd order terms
 and rebuilds output
    if (len(output)<len(database)):
        for k in range(len(tlc)):
            output=sorted(set(output+query(tlc[k])))

    print(" ")
    print('This is our dataset after using our 3rd order features to rebuild the original dataset:')
    print(" ")
    print(output)

    print(" ")
    print('This is our original dataset:')
    print(" ")
    print(database)
    print(" ")
    print('Successful extraction of all usernames in the database')

    return output

def main():
    """Runs your solution -- no need to update (except to maybe change the database)."""
    # Simple implementation of the autocomplete API
    database = ["abracadara", "al", "alice", "alicia", "allen", "alter", "altercation", "bob", "eve",
 "evening", "event", "eventually", "mallory"]
    query = lambda prefix: [d for d in database if d.startswith(prefix)][:5]
    assert extract(query) == database

main()
```

```
Remaining usernames: ['abracadara', 'al', 'alice', 'alicia', 'allen', 'alter', 'altercation', 'bob',
'eve', 'evening', 'event', 'eventually', 'mallory']

This is the set of the first order terms: ['a', 'b', 'e', 'm']

Remaining usernames: ['alter', 'altercation']

Find higher order features

We will filter out [] terms before discovering 2nd order features

This is the post-filtered database: ['abracadara', 'al', 'alice', 'alicia', 'allen', 'alter', 'alter
cation', 'bob', 'eve', 'evening', 'event', 'eventually', 'mallory']


This is the set of the second order terms: ['ab', 'al', 'bo', 'ev', 'ma']

Remaining usernames: ['altercation']

Find higher order terms

[['a', 'b', 'r', 'a', 'c', 'a', 'd', 'a', 'r', 'a'], ['a', 'l'], ['a', 'l', 'i', 'c', 'e'], ['a',
'l', 'i', 'c', 'i', 'a'], ['a', 'l', 'l', 'e', 'n'], ['a', 'l', 't', 'e', 'r'], ['a', 'l', 't', 'e',
'r', 'c', 'a', 't', 'i', 'o', 'n'], ['b', 'o', 'b'], ['e', 'v', 'e'], ['e', 'v', 'e', 'n', 'i', 'n',
'g'], ['e', 'v', 'e', 'n', 't'], ['e', 'v', 'e', 'n', 't', 'u', 'a', 'l', 'l', 'y'], ['m', 'a', 'l',
'l', 'o', 'r', 'y']]


We will filter out [] and ['al'] terms before discovering 3rd order features

This is the post-filtered database: ['abracadara', 'alice', 'alicia', 'allen', 'alter', 'altercatio
n', 'bob', 'eve', 'evening', 'event', 'eventually', 'mallory']


This is the set of the third order terms: ['abr', 'ali', 'all', 'alt', 'bob', 'eve', 'mal']

Remaining usernames: []

Congratulations! We identified all users in our database using 1st, 2nd, and 3rd order features

Our features:
```

```
['a', 'b', 'e', 'm']

['ab', 'al', 'bo', 'ev', 'ma']

['abr', 'ali', 'all', 'alt', 'bob', 'eve', 'mal']

Number of querys to extract features: 16
Number of usernames: 13

Our features:
['a', 'b', 'e', 'm']

['ab', 'al', 'bo', 'ev', 'ma']

['abr', 'ali', 'all', 'alt', 'bob', 'eve', 'mal']


This is our dataset after using our 1st order features to rebuild the original dataset:

['abracadara', 'al', 'alice', 'alicia', 'allen', 'bob', 'eve', 'evening', 'event', 'eventually', 'ma
llory']

This is our dataset after using our 2nd order features to rebuild the original dataset:

['abracadara', 'al', 'alice', 'alicia', 'allen', 'alter', 'bob', 'eve', 'evening', 'event', 'eventua
lly', 'mallory']

This is our dataset after using our 3rd order features to rebuild the original dataset:

['abracadara', 'al', 'alice', 'alicia', 'allen', 'alter', 'altercation', 'bob', 'eve', 'evening', 'e
vent', 'eventually', 'mallory']

This is our original dataset:

['abracadara', 'al', 'alice', 'alicia', 'allen', 'alter', 'altercation', 'bob', 'eve', 'evening', 'e
vent', 'eventually', 'mallory']

Successful extraction of all usernames in the database
```