Computer Science 1 — CSci 1100
Fall Semester, 2019
Final Exam Overview and Practice Questions

SOLUTIONS

The following are the solutions to the practice problems. Please be aware that there may be more than one way to solve a problem and so your answer may be correct despite being different from ours.

Questions

1. Write a version of `merge` that does all of the work inside the `while` loop and does not use the `extend`.

   Solution:

```
def merge2(L1,L2):
    L = []
    i1 = 0
    i2 = 0
    done1 = False
    done2 = False

    while not (done1 and done2):
        if not done1 and (done2 or L1[i1] < L2[i2]):
            L.append(L1[i1])
            i1 += 1
            done1 = i1 >= len(L1)

        else:
            L.append(L2[i2])
            i2 += 1
            done2 = i2 >= len(L2)

    return L
```

2. Using what you learned from writing the solution to the previous problem, write a function to merge three sorted lists. For example

```
print(three_way_merge( [2,3, 4,4, 4, 5], [1, 5, 6,  9], [ 6, 9, 13]))
```

   Should output

```
[1, 2, 3, 4, 4, 4, 5, 5, 6, 6, 9, 9, 13]
```

   Solution:

```
def three_way_merge(L1,L2,L3):
    L = []
    i1 = 0
    i2 = 0
    i3 = 0
    done1 = False
    done2 = False
    done3 = False
```

```
    while not (done1 and done2 and done3):
        if not done1 and (done2 or L1[i1] < L2[i2]) and (done3 or L1[i1] < L3[i3]):
            L.append(L1[i1])
            i1 += 1
            done1 = i1 >= len(L1)

        elif not done2 and (done3 or L2[i2] < L3[i3]):
            L.append(L2[i2])
            i2 += 1
            done2 = i2 >= len(L2)

        else:
            L.append(L3[i3])
            i3 += 1
            done3 = i3 >= len(L3)

    return L
```

3. Given a list of test scores, where the maximum score is 100, write code that prints the number of scores that are in the range 0-9, 10-19, 20-29, ... 80-89, 90-100. Try to think of several ways to do this. Outline test cases you should add.

   For example, given the list of scores

   ```
   scores = [ 12, 90, 100, 52, 56, 76, 92, 83, 39, 77, 73, 70, 80 ]
   ```

   The output should be something like

   ```
   [0,9]: 0
   [10,19]: 1
   [20,29]: 0
   [30,39]: 1
   [40,49]: 0
   [50,59]: 2
   [60,69]: 0
   [70,79]: 4
   [80,89]: 2
   [90,100]: 3
   ```

   Solution: Several different solutions are given, one based on sorting and one based on directly counting the values in each interval, i.e. 0-9, 10-19, 20-29. In either case, we need a bit of special code to handle the last interval because it is 90-100.

   Here is the version that sorts and then scans the list for each interval. The trick is that it only needs to scan once because all the values in each range will be next to each other.

   ```
   scores.sort()
   i = 0
   for d in range(0,9):
       min_i = i                      # remember the first value in this interval
       lower_score = 10*d             # remember the lower bound on the scores for this interval
       upper_score = lower_score + 9  # remember the upper bound
       if upper_score == 99:
           upper_score = 100

       # Count the number of scores and then output
       while i < len(scores) and scores[i] <= upper_score:
   ```

```
        i += 1
    print("[{:d},{:d}]: {:d}" .format(lower_score, upper_score, i-min_i))

# The remaining values will be in the interval 90 to 100
print("[90,100]: {:d}".format(len(scores)-i))

counts = [0] * 10     # form a list of counters
for s in scores:      # assign each score to its appropriate counter
    i = min(9, s//10)  # find the index; using min ensures that i=9 when s = 100
    counts[i] += 1    # increment the counter

# Output
for i in range(0,10):
    lower_score = 10*i
    upper_score = lower_score + 9
    if i == 9:
        upper_score = 100
    print ("[{},{}]: {}".format(lower_score, upper_score, counts[i]))

hist = 10*[0]
bounds = list(range(9, 100, 10))
bounds[-1] = 100
for score in scores:
    for i in range(len(bounds)):
        if score <= bounds[i]:
            hist[i] += 1
            break

val = 0
for i in range(len(bounds)):
    print("[{},{}]: {}".format(val, bounds[i], hist[i]))
    val = bounds[i]+1
```

Test cases:

(a) One should have the value 0 and value 100

(b) Should have examples where there are no values in some of the intervals, particularly 0-9, 90-100 and some other values in between.

(c) Should have some examples where there are several intervals in a row that are empty.

4. Given a list of floating point values containing at least 10 values, how do you find the 10 values that are closest to each other? In other words, find the smallest interval that contains 10 values. By definition the minimum and maximum of this interval will be values in the original list. These two values and the 8 in between constitute the desired answer. This is a bit of a challenging variation on earlier problems from the semester. Start by outlining your approach. Outline the test cases. For example, given the list

```
values = [ 1.2, 5.3, 1.1, 8.7, 9.5, 11.1, 2.5, 3, 12.2, 8.8, 6.9, 7.4,\
      0.1, 7.7, 9.3, 10.1, 17, 1.1 ]
```

The list of the closest 10 should be

```
[6.9, 7.4, 7.7, 8.7, 8.8, 9.3, 9.5, 10.1, 11.1, 12.2]
```

Solution: We start by sorting and then work through the list comparing values that are 9 (not 10) indices apart.

```
values.sort()
first_i = 0
min_dist = values[9] - values[0]
for i in range(1,len(values)-9):
    dist = values[i+9] - values[i]
    if dist < min_dist:
        first_i = i
        min_dist = dist
print(values[first_i:first_i+10])
```

Test cases:

(a) The first ten values are closest.

(b) The last ten values are closest.

(c) There are 10 or more values that are all the same.

5. Consider the following recursive function:

```
def mystery( L, v ):
    if v < len(L):
        x = L[v]
        mystery( L, x )
        print(x)
    else:
        print(v)
```

(a) Show the output from the call:

`mystery( [2, 5, 4, 7, 1], 0 )`

Solution:

```
5
5
1
4
2
```

(b) Write a Python call to the function `mystery` containing a list and an index that causes the recursion to never stop (until the stack overflows). Do this with as short a list as you possibly can.

Solution:

`mystery( [0], 0 )`

(c) Write a Python call to the function `mystery` that causes the program to crash (without the problem of infinite recursion):

Solution:

`mystery( [], -1 )`

6. You are given the following function definition with doctest comments embedded.

```
def f(x, y, z = 0, w = False):
    '''
    >>> f([1,2,3], [4,6,8,5,11], 3, True)
    True
    >>> f([1,2,3,4,5], [0,2,6], -1, True)
    False
```

```
    >>> f([1,2,3,4,5], [2,3,4])
    False
    >>> f([5,4,5,8], [6,7,8,9,10,12], 2, True)
    False
    '''
    count = 0
    for item in x:
        if w and (item+z in y):
            count += 1
    if count == len(x):
        return True
    elif count == len(y):
        return False
```

When the lines below are executed, indicate whether the tests above will pass or fail and, for the failures, indicate what is returned by `f`.

```
import doctest
doctest.testmod()
```

Solution:

```
test1: pass
test2: fail (returns None)
test3: fail (returns None)
test4: fail (returns True)
```

7. You are given a dictionary called `clubs` where each key is the name of a club (a string), and each value is the set of id strings for the students in the club. Write a segment of Python code that creates a set of all student ids (if any) that are in all the clubs in the dictionary.

   Solution:

```
member_lists = list(clubs.values())
in_all = member_lists[0]
for members in member_lists:
    in_all &= members

# or
member_lists = clubs.values()
is_initialized = False
in_all = set()
for members in member_lists:
    if is_initialized:
        in_all &= members
    else:
        in_all = members
        is_initialized = True
```

8. Write a function called `framed` that takes a string storing a name and creates output with a framed, centered greeting. For example, if the name string is `'Tim'` the output should be

```
**********
* Hello! *
*  Tim   *
**********
```

and if the name string is `'Anderson'` the output should be

```
************
*  Hello!  *
* Anderson *
************
```

Solution: Here are two different versions

```
def framed(name):
    n = max( len("Hello!"), len(name) )
    print("*" * (n+4))
    print("* " + "Hello!".center(n) +  " *")
    print("* " + name.center(n) +  " *")
    print("*" * (n+4))

def framed(name):
    s = "Hello!"
    lng = max(len(s),len(name))
    l0 = '*' * (lng+4)
    print(l0)
    r0 = (lng - len(s))//2
    r1 = lng - len(s) - r0
    print('* ' + (' ' * r0) + s + (' ' * r1) + ' *')
    r2 = (lng - len(name))//2
    r3 = lng - len(name) - r2
    print('* ' + (' ' * r2) + name + (' ' * r3) + ' *')
    print(l0)
```

9. Consider a file called `addresses.txt` that contains a name and an address on each line of the file. The parts of each name and address are separated by a `'|'`. For example,

```
John Q. Public |  1313 Mockingbird Way  | Walla Walla, Washington 98765
Samuel Adams  | 1431 Patriot Lane | Apartment 6 |   Washington, Oregon 12345
Dale President | 1600 Pennsylvania Ave. | Washington, D.C. 234781
Thomas Washingon | 153 Washington Street | Anywhere, Anystate 53535
```

Notice there is a variable number of `'|'` in each line, and the string `"Washington"` appears in many places. Fortunately, the input that follows the last `'|'` on each line is in the form `City, State Zip-code`, and there are no commas in the name of the City.

Write a function that is passed the name of a city and returns the number of people stored in `addresses.txt` that live in the given city. For the above version of `addresses.txt` and for the city `Washington`, the function should return the value 2.

Solution:

```
def in_city(city):
    count = 0
    for line in open("addresses.txt"):
        m1 = line.strip().split("|")
        m2 = m1[-1].split(",")
        if city.strip().lower() == m2[0].strip().lower():
            count += 1
    return count
```

10. Write a piece of code that reads from the user an amount between 0 and 1 inclusive, and calculates the number of coins with different value (1 cent, 5 cents, 10 cents, 25 cents) that would be needed to make this amount of cash. Your solution should use the smallest number of coins possible that sum to the number entered. These four coins are the only values you have to worry about and, therefore, you are allowed to "hardcode" these values in. You may also assume that the value entered by the user is a non-negative float and will not have more than two decimal places.

    Three example runs of the program are given below:

```
Please enter an amount between 0 and 1 ==> 0.76
You need ==> 25 cent: 3   10 cent: 0   5 cent: 0   1 cent: 1

Please enter an amount between 0 and 1 ==> 0.81
You need ==> 25 cent: 3   10 cent: 0   5 cent: 1   1 cent: 1

Please enter an amount between 0 and 1 ==> 0.67
You need ==> 25 cent: 2   10 cent: 1   5 cent: 1   1 cent: 2
```

    Solution:

```
# First Solution:
    amt = input("Please enter an amount less than 1 ==> ")
    amt = int(float(amt)*100)
    t25 = amt//25
    amt = amt%25
    t10 = amt//10
    amt = amt%10
    t5 = amt//5
    t1 = amt%5
    print("You need ==> 25 cent: {:d}\t10 cent: {:d}\t 5 cent: {:d}\t1 cent: {:d}".format(t25,t10,t5,t

# Note that because of a rounding a float solution
# like below is inherently risky. On my computer this fails for 0.94
    amt = input("Please enter an amount less than 1 ==> ")
    amt = float(amt)
    q, d, n, p = 0, 0, 0, 0
    c = 0
    while amt > 0.009 and c < 100:
        if amt >= 0.25:
            q += 1
            amt -= 0.25
        elif amt >= 0.10:
            d += 1
            amt -= 0.10
        elif amt >= 0.05:
            n += 1
            amt -= 0.05
        elif amt >= 0.01:
            p += 1
            amt -= 0.01
        c += 1
    print("You need ==> 25 cent: {:d}\t10 cent: {:d}\t 5 cent: {:d}\t1 cent: {:d}".format(q,d,n,c))
```

11. Write a piece of code that asks the user for a string using `input` that has multiple underscore characters and prints the string in "kerned" form such that the extra underscore characters are removed. For example, given the input:

    `___a__b_cd__e f_g___h_`

The program should output (note the space between `e` and `f`):

```
a_b_cd_e f_g_h
```

Remember that `split` returns all strings before and after a given delimiter. For example, `'_a__b_c'.split('_')` returns `['', 'a', '', 'b', 'c']`. As practice, try to solve this using `join` and a list comprehension.
Solution:

```
# First Solution:
s = input("Enter a string ").strip("_")
while s.replace("__","_") != s:
      s = s.replace("__","_")
print(s)

# Second Solution
s = input("Enter a string ").strip("_")
m = s.split("_")
out = ""
for item in m:
   if len(item) > 0:
       out += item + "_"
print(out.strip("_"))

# Third solution
s = input("Enter a string ").strip("_")
s_list = s.strip('_').split('_')
print( '_'.join( [ x for x in s_list if len(x)> 0 ] ))
```

12. Assume the variable `got` contains the characters in a TV show and a score of how much they are loved in the form of a list of lists. Return the name of the most loved character (or characters), i.e. with the highest score in the list `got`. Do not use the `sort` or the `max` functions. Given:

```
got = [ ['tyrion',9], ['cersei',3], ['jon',8], ['daenerys',5], ['arya',9], ['sansa',6], ['tywin',4] ]
```

Your program should print:

```
tyrion arya
```

Solution:

```
# First Solution
m = got[0][1]
for item in got:
    if item[1]>m:
        m = item[1]
for item in got:
    if item[1] == m:
        print(item[0],end=' ')
print()


# Second Solution (More efficient)
top = [ got[0][0] ]
m = got[0][1]

for item in got[1:]:
    if item[1]>m:
```

```
        m = item[1]
        top = [ item[0] ]
    elif item[1] == m:
        top.append( item[0] )
for item in top:
    print(item,end=' ')
print()
```

13. You are given the class `Die` that is defined below:

```
import random

class Die(object):
    def __init__(self, sides):
        sides = int(sides)
        if sides < 1:
            sides = 1
        self.sides = sides
        self.roll()

    def roll(self):
        # random.randint(a, b) returns a random value, x, where a <= x <= b
        self.value = random.randint(1, self.sides)
```

Write a program that uses the `Die` class to create two Die objects. The first die object should be a 6-sided die. The second die object should be a 10-sided die.

Roll the two dice until they both have a value of 1, otherwise known as "snake eyes", and print out the total number of rolls it took for both dice to have a value of 1. Assume that eventually both of the dice will have a value of 1, terminating your program.

Solution:

```
d1 = Die(6)
d2 = Die(10)

rolls = 1 # or 0
while True:
    if d1.value == d2.value and d2.value == 1:
        print("Rolls:", rolls)
        break
    d1.roll()
    d2.roll()
    rolls += 1
```

14. Write a function that reads integers from a file (call it `integers.txt`), storing them in a list. It should stop and return the list when there are no more integers or when a negative integer is found. Each input line contains just one integer.

Solution:

```
def read_them():
    result = []
    for line in open('integers.txt'):
        value = int(line)
        if value < 0:
            break
        result.append(value)
    return result
```

15. Write a function that takes as input an unordered list, and returns the number of values that would remain in the list after all duplicates are removed. Examples:

```
>>> find_dup([1, 5, 3, 6, 6, 3, 1])
4
>>> find_dup([3, 1, 2, 4])
4
>>> find_dup([2, 1, 2, 1, 1, 2, 2])
2
>>> find_dup([])
0
```

Note that this problem is very easy to solve using sets, but you can solve it without sets as well. Try to come up with both solutions.

Solution: Here is the short solution using a set and the `len` function

```
def find_dup(L):
    return len(set(L))
```

Here is a longer solution that counts duplicates and returns the difference with the length of the list

```
def find_dup(L):
    L1 = sorted(L)
    dup = 0
    for i in range(len(L1)-1):
        if L1[i] == L1[i+1]:
            dup += 1
    return len(L) - dup
```

Here is a slightly different solution that counts distinct values directly

```
def find_dup(L):
    L1 = sorted(L)
    if len(L) == 0:
        distinct = 0
    else:
        distinct = 1
    for i in range(len(L1)-1):
        if L1[i] != L1[i+1]:
            distinct += 1
    return distinct
```

Finally, here is a double for loop version. Although it works, the other solutions are much better.

```
def find_dup(L):
    if len(L) == 0:
        count = 0
    else:
        count = 1
    for i in range(len(L)-1):
        is_distinct = True
        for j in range(i+1,len(L)):
            if L[i] == L[j]:
                is_distinct = False
                break    # out of inner loop. not strictly necessary
        if is_distinct:
            count += 1
    return count
```

16. Write a function called `notused` that takes a list of words as its single parameter, and returns a set containing the letters of the English alphabet that are not used by any of the words in the input list. Your function must use sets. Here is an example of how your function should work:

```
>>> notused([ "Dog", "pony", "elephant", "Tiger", "onyx", "Zebu" ])
set(['c', 'f', 'k', 'j', 'm', 'q', 's', 'w', 'v'])
```

Hint: you can use the following set in your solution:

```
all_letters = set("abcdefghijklmnopqrstuvwxyz")
```

Solution:

```
def notused(words):
    all_letters = set("abcdefghijklmnopqrstuvwxyz")
    all_used = set()
    for word in words:
        all_used = all_used | set(word.lower())
    return all_letters - all_used
```

17. In the iterative version of merge sort we discussed in class, the code starts by dividing the list to be sorted into a list of lists, each of length 1. The actual Python sort function starts instead by looking for "runs" — consecutive values in the list that are already in order. Each run is an initial list for merge sort. For example, the list

```
L = [ 15, 3, 6, 19, -1, 7, 9, 3, 5 ]
```

would be divided into the list of lists

```
lists = [ [15], [3,6,19], [-1,7,9], [3,5] ]
```

Write a function called `runs` that takes `L` as an argument and returns `lists`.

Solution

```
def runs(L):
    # We will add each run to the end
    runs = []

    # Handle the special case of an empty list
    if len(L) == 0:
        return runs

    # Keep a list call new_run that is added to whenever the next
    # values (v) is greater than the previous value.  When this is not
    # the case, the end of a run has been found so add to the runs
    # list and start a new run.
    new_run = [ L[0] ]
    for v in L[1:]:
        if v >= new_run[-1]:
            new_run.append(v)
        else:
            runs.append(new_run)
            new_run = [ v ]

    # Need to save the last run before returning
    runs.append( new_run )
    return runs
```

18. Write a version of binary search called *trinary search* where the search interval is split into thirds instead of in half. As with binary search, this function should return the index of the location where the first value `x` either is stored in the list or where it should be inserted if `x` is not there. Before starting on this, please make sure you can do the hand simulations of binary search so that you understand the importance of `low`, `mid` and `high` in the code.

Solution:

```
def trinary_search( x, L):
    low = 0
    high = len(L)
    while low != high:
        mid0 = low + (high-low)//3
        mid1 = low + 2*(high-low)//3
        if x > L[mid1]:
            low = mid1+1
        elif x > L[mid0]:
            low = mid0+1
            high = mid1
        else:
            high = mid0
    return low
```

19. (This problem was essentially given as a lecture exercise, but it had appeared on an earlier exam.) Below you will find the merge function from class. Show how to modify it so that when a value that appears in both lists `L1` and `L2` it only appears once in `L`. You may assume that there are no duplicate values in `L1` and no duplicate values in `L2`. As an example, if

```
L1 = [ 1, 3, 5, 6, 7, 8, 10 ]
L2 = [ 5, 6, 10, 13, 14 ]
```

then after the merge

```
L = [ 1, 3, 5, 6, 7, 8, 10, 13, 14 ]
```

```
def merge(L1,L2):
    i1 = 0
    i2 = 0
    L = []
    while i1<len(L1) and i2<len(L2):
        if L1[i1] < L2[i2]:
            L.append(L1[i1])
            i1 += 1
        else:
            L.append(L2[i2])
            i2 += 1
    L.extend(L1[i1:])
    L.extend(L2[i2:])
    return L
```

```
def merge(L1,L2):
```

Solution: All of the modifications occur within the while loop. Here is the complete function

```
def merge(L1,L2):
    i1 = 0
    i2 = 0
```

```
    L = []
    while i1<len(L1) and i2<len(L2):
        if L1[i1] < L2[i2]:
            L.append(L1[i1])
            i1 += 1
        elif L1[i1] > L2[i2]:
            L.append(L2[i2])
            i2 += 1
        else:
            L.append(L2[i2])
            i1 += 1
            i2 += 1
    L.extend(L1[i1:])
    L.extend(L2[i2:])
    return L
```

20. The list sort function takes an optional comparison function as an argument. For example, after the following code list `L` is in *decreasing order*.

```
def rev(x):
    return -x
L = [1, 10, 3, 6]
L.sort(key=rev)
print(L)
```

Would print

```
[10, 6, 3, 1]
```

In order to achieve this, the comparison function negates the key value.

Write a function `cmp_string` that can be used to order a list of words (comprised of only lower case letters) primarily by length. Words that are the same length should be in alphabetical order. For example, a correct version of this function should result in the following output

```
>>> L = [ "apple", "ball", "car", "card", "basket", "car", "honey" ]
>>> L.sort(key=cmp_string)
>>> print(L)
['car', 'car', 'ball', 'card', 'apple', 'honey', 'basket']
```

Your key function will need to return a tuple.

Solution:

```
def cmp_string(s):
    return len(s), s
```

21. In order to sort a list of 2d point coordinates, `[x,y]`, the Python sort function sorts by the first, `x`, coordinate in the list, and then breaks any ties by looking at the second, `y`, coordinate. If you want instead to sort by the `y` coordinate and then by the `x` coordinate, you need to specify an alternate `key` function to the sort. Write two different `key` routines – the first, called `return_y` that given a 2 element list returns just the `y` coordinate and a second called `flip_coordinates()` that given a 2 element list returns a new list with the `x` and `y` coordinates swapped. Now using each of these key functions, write code to sort a list by the `y` coordinate first and then the `x` coordinate. Note that for the `return_y()` function you will need to take advantage of the **stable sort** property of the Python sort.

As examples,

```
print return_y( [1,3])   # outputs 3
print return_y( ([1,6])   # outputs 6
print flip_coordinates( [1,3])   # outputs [3,1]
print flip_coordinates( [1,6])   # outputs [6,1]
```

Using either of your sorts, the list

```
L = [ [11,2], [5,8], [5,2], [12,3], [1,3], [10,2], [12,1], [12,3] ]
```

will be sorted as:

```
... Sort Code ...
print(L)

[[12, 1], [5, 2], [10, 2], [11, 2], [1, 3], [12, 3], [12, 3], [5, 8]]
```

Solution:
Using `return_y`:

```
def return_y(x):
    return x[1]

L = [ [11,2], [5,8], [5,2], [12,3], [1,3], [10,2], [12,1], [12,3] ]
L.sort()
L.sort(key=return_y)
print(L)
```

or Using `flip_coordinates()`:

```
def flip_coordinates(x):
    return [x[1], x[0]]

L = [ [11,2], [5,8], [5,2], [12,3], [1,3], [10,2], [12,1], [12,3] ]
L.sort(key=flip_coordinates)
print(L)
```

Now rewrite your solution to the previous question to eliminate `return_y` and `flip_coordinates()` by using lambda functions. **Solution**:

```
L = [ [11,2], [5,8], [5,2], [12,3], [1,3], [10,2], [12,1], [12,3] ]
L.sort()
L.sort(key=lambda x: x[1])
print(L)
```

or

```
L = [ [11,2], [5,8], [5,2], [12,3], [1,3], [10,2], [12,1], [12,3] ]
L.sort(key=lambda x: [x[1],x[0]])
print(L)
```

22. Given a list of point coordinates, write a function that converts from cartesian `(x, y)` to polar `(angle, radius)`, then use `map` to covert each entry in a list of points from cartesian to polar. Note that to get angle in degrees you need to calculate `atan2(y, x) * 180 / p`.

```
points = [(-1, 1), (3, 4), (2, -3)]
```

your code would generate:

```
[(135.0, 1.4142135623730951), (53.13010235415598, 5.0), (-56.309932474020215, 3.605551275463989)]
```

Solution:

```
from math import atan2, pi
def cartesian_to_polar(pt):
    return (atan2(pt[1], pt[0]) * 180 / pi, (pt[0]**2 + pt[1]**2)**0.5)

list(map(cartesian_to_polar, points))
```

Now replace the cartesian_to_polar function with a `lambda` function to generate the same answer, and then use a `list comprehension` to get the same result.

Solution:

```
list(map(lambda pt:(atan2(pt[1], pt[0]) * 180 / pi, (pt[0]**2 + pt[1]**2) ** 0.5), points))
[(atan2(pt[1], pt[0])*180/pi, (pt[0]**2+pt[1]**2)**0.5) for pt in points]
```

23. Given a list of words as strings, use `filter` and a `lambda` function to return all of the words that contain the string `'ue'`. For example, given:

```
words = ['python', 'queue', 'blue', 'coconut', 'true', 'grail']
```

your code would generate:

```
['queue', 'blue', 'true']
```

Solution:

```
 list(filter(lambda word: 'ue' in word, words))
```

Then use a `list comprehension` to get the same result.

Solution:

```
[word for word in words if 'ue' in word]
```

24. Write a recursive function to generate the greatest common denominator of two numbers using Euclid's algorithm which says that: Given two numbers `x` and `y` with `x > y`, return `y` if `y` divides `x`, otherwise, calculate the greatest common deniminator of `y` with the remainder of `x / y`. **Solution**:

```
def gcd(x, y):
    z = x % y
    if z == 0:
        return y
    else:
        return gcd(y, z)
```

Once you have the recursive version written, rewrite it not using recursion.

Solution:

```
def gcd_iter(x, y):
    z = x % y
    while z != 0:
        x = y
        y = z
        z = x % y
    return y
```

25. Given the following code, what is the output of the call `x(3)`? What is the base case?

```
def x(y):
    if y < 10:
        print(y*'=')
        x(y+1)
        print(y*'+')
    else:
        print(y*'*')
```

Solution:

```
===
====
=====
======
=======
========
=========
**********
+++++++++
++++++++
+++++++
++++++
+++++
++++
+++
```

The base case is `y>=10`