Computer Science 1 — CSci 1100 — Fall 2017
Final Exam
December 18, 2017

SOLUTIONS

1. (**12 points**) Show the output of the following:

Code:

```
# Part (a)

L=['Terry','Eric','Terry','Michael']
print (sorted(list(
      filter(lambda x: x.find('r') != -1, L))))
A = [(L[i],L[j])
      for i in range(len(L))
         for j in range(len(L)-1,i,-2)]
print (A)
```

Solution:

```
['Eric', 'Terry', 'Terry']
[('Terry', 'Michael'), ('Terry', 'Eric'),
   ('Eric', 'Michael'), ('Terry', 'Michael')]
```

Code:

```
# Part (b)
D = {'Rick': {'Scientist', 'Genius', 'Dad'}, \
     'Beth': {'Daughter', 'Surgeon', 'Mother'}, \
     'Summer': {'Daughter'}}
E = {}
for k in D:
    for i in D[k]:
        if not i in E:
            E[i] = set()
        E[i].add(k)
F = sorted(E.keys())
print(F[0])
print(len(E))
print(sorted(E['Genius']))
print(sorted(E['Daughter']))
```

Solution:

```
Dad
6
['Rick']
['Beth', 'Summer']
```

Code:

```
# Part (c)
def e(s):
    if len(s) == 0:
        return
    print(s[0], end='')
    e(s[1:])
    print(s[0], end='')

e("pul")
print()
e(list(range(2)))
print()
```

Solution:

```
pullup
0110
```

2. (**12 points**) For each of the following write one or **a few lines of Python code** to solve the problem. Assume you are typing the lines into the Python shell. You may use any combination of the techniques we have discussed throughout the semester, including list comprehensions, map, filter and lambda functions. However, any use of a `for` loop or an `if` (except within a list comprehension), or a `while` loop will receive a 0. Solutions of more than 4 lines will also receive a 0.

Code:	**Solution:**

(a) Given a list `L`, write code to print the strings in it formatted with parentheses as shown below. As an example if
`L=[('Porg','Ahch-To'),`
`('Chewbacca','Kashyyyk'),`
`('Vulptex','Crait')]`
the code should print
`Porg (Ahch-To)`
`Chewbacca (Kashyyyk)`
`Vulptex (Crait)`
	`print ('\n'.join(map (lambda x: '{}`
	`({})'.format(x[0],x[1]), L)))`

(b) Given a list `L` of integer and float values, output a list of strings where each component is a string of `n` `'*'s` for `n` an integer value in `L`. You will need to filter all float values from `L` before generating the list of `'*'s`.Remember that `type(x)` returns `int` or `float` depending on the type of `x`. For example, if
`L=[4, 3, 3.1, 1.0, 2]`
the output would be
`['****', '***', '**']`
	Using list comprehension
	`# print is optional`
	`print([i*'*' for i in L if type(i)==int])`
	Using map / filter
	`# print is optional`
	`print(list(map(lambda x: x*'*', \`
	`filter(lambda x: type(x)==int, L))))[x[0]*x[1] \`
	`for x in L if x[1] > 0]`

(c) Given 4 sets, `i1`, `i2`, `j1` and `j2`, output a new set that contains all the values that occur in `i1` and `i2` or in `j1` and `j2`, but that do not occur in all 4 sets. As an example, if
`i1=set([1, 3, 5, 6])`
`i2=set([1, 5, 9, 13])`
`j1=set([1, 2, 3, 4])`
`j2=set([1, 2, 4, 6, 8])`
then the output should be (note the actual printed ordering may be different as this is a set):
`{2, 4, 5}`
	`# print is optional`
	`print((i1&i2)^(j1&j2))`
	or
	`# print is optional`
	`print(((i1&i2)|(j1&j2))-(i1&i2&j1&j2))`

3. (**12 points**) Suppose you are given a file, `trees.txt` that contains information about "elm" and "birch" trees. We want to divide this file into two different files named `elms.txt` and `birches.txt`. Write a program to read in `trees.txt` and write any line that has the word `elm` in it into the file `elms.txt`, and every line that has the word `birch` in it into the file `birches.txt`. Lines that do not have either tree in them, or lines that have both trees in them, should just be discarded. Be careful about capitalization.

   Given the following lines in `trees.txt`,

```
The elm is stately
Birch trees are lovely
Birch trees and elm trees have leaves
Leaves are green
Elm is very nice wood
```

   The file `elms.txt` would have:

```
The elm is stately
Elm is very nice wood
```

   and `birches.txt` would have:

```
Birch trees are lovely
```

   Solution:

```
elms = open('elms.txt','w')
birches = open('birches.txt','w')
for line in open('trees.txt'):
    if line.lower().find('elm') != -1 and line.lower().find('birch') == -1:
        elms.write(line)
    elif line.lower().find('elm') == -1 and line.lower().find('birch') != -1:
        birches.write(line)
elms.close()
birches.close()
```

4. (**10 points**) Write a function that takes as input a list of positive numbers and returns the value stored in the highest peak. A peak is a value in the list that is larger than the values immediately before and after it. Your function should find and return the largest peak value. If there are no peaks in L, your function should return -1.

   The following are example runs of this function:

```
>>> highest_peak([2,5,4,1,6,6,8,7])
8
>>> highest_peak([2,5,4,2,5,6])
5
>>> highest_peak([1,2,4,5])
-1
```

Solution:

```
def highest_peak(L):
    v = -1
    for i in range(1, len(L)-1):
        if L[i] > L[i-1] and L[i] > L[i+1]:
            v = max(v, L[i])
    return v
```

5. (**12 points**) Suppose you are given a dictionary `prefs` containing movie preferences of a group of friends such as the following :

```
prefs = {'Ramona': ['Star Wars', 'Thor', 'Disaster Artist', 'Justice League'],
         'Scott' : ['Disaster Artist', 'Lady Bird', 'Star Wars'],
         'Knives': ['Justice League', 'Thor', 'Lady Bird', 'Disaster Artist'] }
```

Each key in `prefs` is the name of a friend and each value gives the names of movies listed in the order of preference for this friend, with the first movie being the most preferred.

Each movie gets a score according its location in these preferences, summed over all the lists. For example, `Disaster Artist` gets `3+1+4=8` points because it appears in 3rd place for `Ramona`, 1st place for `Scott` and 4th place for `Knives`.

Assuming such a dictionary is already stored in your program, write a program that computes the total score for each movie and prints the movie (or movies) with the lowest score (most preferred) in alphabetical order.

For example, if you were given the dictionary above, your program would output the following:

```
Star Wars: score 4
Thor: score 4
```

Solution:

```
scores = {}
for key in prefs:
    for i in range(len(prefs[key])):
        movie = prefs[key][i]
            if movie not in scores:
                scores[movie] = i+1
            else:
                scores[movie] += i+1

max_score = min(scores.values())
for name in sorted(scores):
    if scores[name] == max_score:
        print ("{}: score {}".format(name, scores[name]))
```

6. (**12 points**) Suppose you are given the following class definition in a file called `Pacman.py`:

```
class Pacman(object):
    def __init__(self, name, x=0,y=0):
        self.name, self.x, self.y = name,x,y
    def __str__(self):
        return "{} @ ({},{})".format(self.name, self.x,self.y)
    def move(self, dir):
        if dir == 'up':
            self.y -= 1
        elif dir == 'down':
            self.y += 1
        elif dir == 'left':
            self.x -= 1
        elif dir == 'right':
            self.x += 1
```

Suppose a file called `movepacman.txt` contains the name and initial location of a Pacman object in the first line, followed by a number of movements. The file may have empty lines after the first line which should be skipped.

Write a program that reads this file to create a Pacman object, prints its initial location, moves it according to the instructions in the file and then prints its final location. You must use all the methods given in this class in your implementation correctly.

For example, if you were given the following contents of the file `movepacman.txt`:

```
Blinky 20 20
up
LEFT
Right

right
doWn
```

your program would output the following:

```
Initial location: Blinky @ (20,20)
After 5 moves Blinky @ (21,20)
```

Solution:

```
f = open('file1.txt')
a = f.readline().split()
p = PacMan(a[0], int(a[1]), int(a[2]))

print ("Initial location:", str(p))
lineno = 0
for line in f:
    if len(line.strip()) != 0:
        lineno += 1
        p.move(line.strip().lower())
print ("After", lineno, "moves", str(p))
```

7. (**12 points**) Write a program that reads two strings from the user and then prints them out in a crossword pattern. The program should print out the first word horizontally and the second word vertically with the two words "crossing" at the location of the first shared letter in string 1 and its first occurrence in string 2. You can assume that there is always a shared letter. For example, if the program is given the words `cross`, and `word`, then the first shared letter is "r". Running the program would give:

```
String 1: cross
String 2: word
 w
 o
cross
 d
```

Solution:

```
string1 = input("String 1: ").strip()
string2 = input("String 2: ").strip()
for index1 in range(len(string1)):
    index2 = string2.find(string1[index1])
    if index2 != -1:
        break
for i in range(index2):
    print(index1*' '+string2[i])
print(string1)
for i in range(index2+1,len(string2)):
    print(index1*' '+string2[i])
```

8. (**16 points**) Consider the binary search algorithm discussed in class (also given below).

```
def binary_search(x, L):
    low = 0
    high = len(L)
    while low != high:
        mid = (low+high)//2
        if x > L[mid]:
            low = mid+1
        else:
            high = mid
    return low
```

(a) (12/16 pts) Write a function `insert_value(x, L)` that takes a value `x` and a list `L` and inserts `x` into `L` in numerical order if only if `x` does not already exist.

Your insert function must use the `binary_search()` algorithm to determine where to insert `x`. It must not use the list `insert()` function; however it can use the list `append()` function. Be extra careful about adding values to the end of the list.

For example, given:

```
L = [ 1, 7, 11, 15, 16, 17, 19 ]
insert_value( 11, L )
print('1:', L)
insert_value( 18, L )
print('2:',L)
insert_value( -1, L)
print('3:', L)
insert_value( 25, L)
print('4:',L)
```

will give:

```
1: [1, 7, 11, 15, 16, 17, 19]
2: [1, 7, 11, 15, 16, 17, 18, 19]
3: [-1, 1, 7, 11, 15, 16, 17, 18, 19]
4: [-1, 1, 7, 11, 15, 16, 17, 18, 19, 25]
```

Solution:

```
def insert_value(x, L):
    where = binary_search(x, L)
    if where == len(L) or L[where] != x:
        L.append(x)
        for ctr in range(len(L)-1, where, -1):
            L[ctr], L[ctr-1] = L[ctr-1], L[ctr]
```

Answer box for this question is on the next page. You can use this space for scratch work.

Write your answer for 8(a) within the box below.

Part 8a answer:

(b) (4/16pts) Much like we can rewrite the `merge_sort()` algorithm to be recursive, we can also rewrite the `binary_search()` algorithm to be recursive. The following framework sets up the base cases. Add the recursive cases to complete the routine.

Note that you will need one recursive call when the value falls in the lower half of the list and a second when it falls in the upper half of the list. Be sure to manage your return index correctly.

```
def binary_search(x, L):
    mid = len(L)//2
    if len(L) == 0:
        return 0
    elif len(L) == 1:
        if x <= L[0]:
            return 0
        else:
            return 1
    '''
    Insert the recursive cases here.
    '''
```

Solution:

```
def binary_search(x, L):
    mid = len(L)//2
    if len(L) == 0:
        return 0
    elif len(L) == 1:
        if x <= L[0]:
            return 0
        else:
            return 1
    elif x < L[mid]:
        return binary_search(x, L[:mid])
    else:
        return mid + binary_search(x, L[mid:])
```

9. (**2 points**) Please read carefully and respond below:

   (a) All grades except Lab 12, HW 8 and the final exam are frozen and unchangeable.

   (b) Grades for Lab 12 and HW 8 will be frozen and unchangeable after Tuesday, 12/19 at 5 pm EST. Before that time you must check your Rainbow grades for correctness and appeal any issues with Lab 12 grades (to your lab TA) and HW 8 grading (to the TA who graded).

   (c) Sometime before Wednesday 12/20 at 12 noon EDT Gradescope grades will be released for the final exam. These grades will have already been checked carefully. You have until Thursday 12/21 at 9 am to request regrades. After that time all final exam grades will be be frozen.

   (d) Final grades for the course will be completed and posted by 5 pm Thursday 12/21.

   (e) Please write **True** (2 points) in the box below to indicate that you have read and understood these instructions.

Congratulations! CS-1 is over. Go forth, have some fun, and enjoy the Break!
Remember: We are always looking for good mentors! Contact your lab TA if you are interested!

This part is completely optional and will not affect your grade: Use the space below to draw art work to entertain us (and make you famous) during final exam grading.