Computer Science 1 — CSci 1100 — Spring 2017
Final Exam
May 8, 2017

SOLUTIONS

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

Code:	**Solution:**

`# Part (a)`
`S = "face12"`
`T = ','.join(list(S))`
`U = T.replace('f', '15').replace('a','10').\`
`  replace('c','12').replace('e','14').split(',')`
`x = 7`
`v = [x+int(i) for i in U]`
`w = list(filter(lambda x: int(x) >= 10, U))`
`print(T)`
`print(U)`
`print(v)`
`print(w)`
	`f,a,c,e,1,2`
	`['15', '10', '12', '14', '1', '2']`
	`[22, 17, 19, 21, 8, 9]`
	`['15', '10', '12', '14']`

`# Part (b)`
`def print_T(t):`
`  if len(t) == 1:`
`  print(t[0])`
`  else:`
`  print_T(t[1])`
`  print(t[0])`
`  print_T(t[2])`
 	
`T = [1, [2, [3], [4]], [5, [6], [7]]]`
`print_T(T)`
	`3`
	`2`
	`4`
	`1`
	`6`
	`5`
	`7`

`# Part (c)`
`L = sorted(set([3, 9, 5, 2, 5, 2, 3, 3]))`
`D = {5:['car', 'truck', 'van'], \`
`3:['pig', 'cow', 'horse']}`
`for val in L:`
`  if val in D.keys() and len(D[val]) == val:`
`  print(val, sorted(D[val]))`
`  elif val in D:`
`  print(D[val])`
`  else:`
`  print(val)`
	`2`
	`3 ['cow', 'horse', 'pig']`
	`['car', 'truck', 'van']`
	`9`

2. (**12 points**) For each of the following write **one or two lines of Python code** to solve the problem. (All problems can be solved in one line, but it may be easier and clearer to write two.) Assume you are typing the lines into the Python shell. You may use any combination of techniques we have discussed throughout the semester, including list comprehensions, map, filter and lambda functions. However, any use of a `for` loop (except within a list comprehension) or a `while` loop will result in a 0 for that answer.

Code:	**Solution:**

(a) Given sets `s1`, `s2`, `s3`, and `s4`, create a new set `t` that contains all values that are in **both** `s1` and `s2` but not in both `s3` and `s4`. For example, given
`s1 = set(['red','black','blue','green'])`
`s2 = set(['red','yellow','blue','black'])`
`s3 = set(['straw','blue','black','goose'])`
`s4 = set(['straw','blue','goose','rasp'])`
then after your code `t` should be
`{'red', 'black'}`
	`t = (s1&s2) - (s3&s4)`
	or
	`t =`
	`  s1.intersection(s2).difference(s3.intersection(s4))`

(b) Given a list `L` of tuples where each tuple has an animal name as the first component and the number of legs as the second component, return a new list of strings where each string is just the name of the animal repeated based on its number of legs. Animals with less than 1 leg should be filtered out from the list. Note that `0*string` for any `string` returns the empty string `""`. This is not the same as filtering the animal. For example, if
`L=[("oldpegleg", 1), ("cow", 4), ("trout", 0),\`
`  ("jellyfish", 0),  ("student", 2)]`
the returned list would be
`['oldpegleg', 'cowcowcowcow', 'studentstudent']`
	Using list comprehension
	`[x[0]*x[1] for x in L if x[1] > 0]`
	Using map / filter
	`list(map(lambda a: a[0]*a[1], \`
	`filter(lambda a: a[1]>0, L)))`

(c) Given two lists `L1` and `L2` where `L1` is a list of integers and `L2` is a list of strings, create a new list `L3` that contains `True` if L2[i] has the exact number of "e"s in L1[i] and `False` otherwise. For example, if
`L1 = [2, 1, 3, 0]`
`L2 = ['eel', 'are', 'tree', 'piece']`
then `L3` should be:
`[True, True, False, False]`
You can assume `L1` only has valid indices for list `L2`
	`[L2[i].count('e')==L1[i] for i in range(len(L1))]`
	or
	`list(map(lambda i: L2[i].count('e')==L1[i], \`
	`range(len(L1))))`

3. (**12 points**) Write a program that reads a string from the user and then writes out a block of `len(string)` lines where the string rotates one character to the left on every successive line. For example, if the program is given `"Monty Python"`, you would have the following output. Note that a space is printed between every character on the line.

```
Enter a string: Monty Python

M o n t y   P y t h o n
o n t y   P y t h o n M
n t y   P y t h o n M o
t y   P y t h o n M o n
y   P y t h o n M o n t
  P y t h o n M o n t y
P y t h o n M o n t y
y t h o n M o n t y   P
t h o n M o n t y   P y
h o n M o n t y   P y t
o n M o n t y   P y t h
n M o n t y   P y t h o
```

Solution:

```
string = input("Enter a string: ")
print()
letters = list(string)
for ctr in range(len(letters)):
    for letter in letters:
        print(letter, end=" ")
    print()
    rotate = letters.pop(0)
    letters.append(rotate)
```

or

```
string = input("Enter a string: ")
print()
letters = list(string)
for ctr in range(len(letters)):
    for index in range(ctr, len(letters)):
        print(letters[index], end=" ")
    for index in range(ctr):
        print(letters[index], end=" ")
    print()
```

4. (**14 points**) Write a function `find_factors` that takes an integer parameter `val`. If `val<=0` your program should return nothing (**None**). Otherwise, your function should create a list `L` of tuples containing the factors of `val`. I.e. each tuple contains 2 integer values `x` and `y` such that `x * y = val`. Note that if `(2, 3)` is a tuple in the list, then you should not include the tuple `(3, 2)` as it is redundant. Once your function is written, write a short program that reads an integer `value` from the user, calls `find_factors` on `value` and prints out the answer.

   Here are two example runs of the program:

```
Enter an integer: 962

The factors of 962 are:
[(1, 962), (2, 481), (13, 74), (26, 37)]
```

   and

```
Enter an integer: 0

The factors of 0 are:
None
```

Solution:

```python
def find_factors(val):
    if val <= 0:
        return
    L = []
    for i in range(1, int(val**0.5) + 1):
        if val % i == 0:
            L.append((i, val//i))
    return L

if __name__ == "__main__":
    val = int(input("Enter an integer: ").strip())
    factors = find_factors(val)
    print()
    print("The factors of {} are:\n{}".format(val, factors))
```

5. (**16 points**) A common problem in medical research is the need to make subjects "anonymous" by replacing specific information such as name, date of birth, dates of procedures, etc. with placeholder information. Of course, if you ever need to get the original data back, you need to have a method for returning the data to its original state. Assume you have a dictionary of dictionaries:

```
p1={"John Cleese":"Luke Skywalker", "01/10/2006":"01/01/1800",  "02/19/2008":"02/10/1802"}
p2={"Michael Palin":"Jabba the Hut", "09/19/2010":"01/01/1800",  "10/27/2015":"02/09/1815"}
p3={"Eric Idle":"Han Solo",  "07/16/1995":"01/01/1800"}
patients = {"PAT1901": p1, "PAT8235": p2, "PAT9657": p3}
```

The key to the dictionary is a case identifier, `"PAT1901", "PAT8235",` or `"PAT9657"` and the value is a dictionary of substitution codes.

Assume that you already have the patients dictionary. Write code to ask the user for a source filename and an anonymization file name. The source file will have a case identifier as the first line in the file followed by some patient information. Read the file and use the case identifier to replace all occurrences of patient information with the appropriate substitution code. For example, if the first line of the file has `PAT9657`, then in the rest of the file, "Eric Idle" should be replaced by "Han Solo" and all occurrences of "07/16/1995" should be replaced by "01/01/1800". The result should be written out to the anonymization file. So if the input file has:

```
PAT8235
Patient: Michael Palin,
Date of diagnosis: 09/19/2010
Date of procedure: 10/27/2015

Michael Palin was diagnosed with
terminal funnyness on 09/19/2010. After
a successful funnybone-ectomy on 10/27/2015
Michael Palin is now living a normal life.
```

Then the anonymization file would have

```
PAT8235
Patient: Jabba the Hut,
Date of diagnosis: 01/01/1800
Date of procedure: 02/09/1815

Jabba the Hut was diagnosed with
terminal funnyness on 01/01/1800. After
a successful funnybone-ectomy on 02/09/1815
Jabba the Hut is now living a normal life.
```

You can assume capitalization and spelling are correct and that the input file exists. Note that you cannot assume any specific structure for the input file other than the first line will be a valid case identifier.

WRITE YOUR SOLUTION ON THE NEXT PAGE

Solution: Here is a first solution that creates a dictionary of sets.

```
inputfile = input("Input file: ").strip()
outputfile = input("Output file: ").strip()
infile = open(inputfile)
record = infile.readline().strip()
patient = patients[record]
rest = infile.read()
for key in patient:
    rest = rest.replace(key, patient[key])
outfile = open(outputfile, 'w')
outfile.write(record + '\n')
outfile.write(rest + '\n')
outfile.close()
```

6. (**16 points**) Think back to our sudoku lab. We represented the sudoku board as a list of lists:

```
bd = [ [ '1', '.', '.', '.', '2', '.', '.', '3', '7'],
       [ '.', '6', '.', '.', '.', '5', '1', '4', '.'],
       [ '.', '5', '.', '.', '.', '.', '.', '2', '9'],
       [ '.', '.', '.', '9', '.', '.', '4', '.', '.'],
       [ '.', '.', '4', '1', '.', '3', '7', '.', '.'],
       [ '.', '.', '1', '.', '.', '4', '.', '.', '.'],
       [ '4', '3', '.', '.', '.', '.', '.', '1', '.'],
       [ '.', '1', '7', '5', '.', '.', '.', '8', '.'],
       [ '2', '8', '.', '.', '4', '.', '.', '.', '6'] ]
```

with `'.'` representing a place that has not yet been assigned a number. Create a `Sudoku` class with an attribute that holds our list of lists and an initialization method. The class should have an attribute `board` representing the board as a list of lists and the initialization method should take a list of tuples where the first value in the tuple is the row coordinate, the second value is the column coordinate and the third value is the string to put in the sudoku board. Creating a sudoku with no list should create an empty board. For example,

```
a = Sudoku()
b = Sudoku([(1, 2, '7'), (2, 5, '9')])
```

would result in `a.board` having the value:

```
[[ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 [ '.', '.', '.', '.', '.', '.', '.', '.', '.']]
```

and with `b.board` having the value:

```
[['.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '7', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '9', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '.', '.', '.', '.'],\
 ['.', '.', '.', '.', '.', '.', '.', '.', '.']]
```

Be careful about aliasing.

WRITE YOUR SOLUTION ON THE NEXT PAGE

Solution:

```
class Sudoku(object):
    def __init__(self, L=[]):
        line = 9*['.']
        self.board = []
        for i in range(9):
            self.board.append(line.copy())
        for tup in L:
            self.board[tup[0]][tup[1]] = tup[2]
```

or

```
class Sudoku(object):
    def __init__(self, L=[]):
        line = []
        for i in range(9):
            line.append('.')
        self.board = []
        for i in range(9):
            self.board.append(line.copy())
        for tup in L:
            self.board[tup[0]][tup[1]] = tup[2]
```

7. (**12 points**) Wooden boards are categorized according to their length and width. Boards greater than or equal to 4 feet in length are considered long and boards less than 4 feet are considered short. Boards greater than or equal to 5 inches in width are considered wide and boards less than 5 inches are considered narrow. Boards that are long and wide are "planks". Boards that are long and narrow are "poles". Boards that are short and wide are "shingles". Boards that are short and narrow are "sticks". Write a program to repeatedly ask the user for the dimensions of a board and count how many of each type are entered. The user will enter length in feet and width in inches on the same line, separated by a comma. When the user enters a blank line, print out the totals for each type and terminate the program. You can assume that the input is valid. Below is a sample run,

```
Enter the dimensions (l, w): 4, 5
Enter the dimensions (l, w): 5, 6
Enter the dimensions (l, w): 6, 7
Enter the dimensions (l, w): 6, 2
Enter the dimensions (l, w): 6, 1
Enter the dimensions (l, w): 3, 7
Enter the dimensions (l, w): 3, 1
Enter the dimensions (l, w):
There were 3 planks, 2 poles, 1 shingles, and 1 sticks
```

Solution:

```
planks = 0
poles = 0
shingles = 0
sticks = 0
while True:
    dims = input("Enter the dimensions (l, w): ")
    if dims.strip() == "":
        break
    l, w = dims.strip().split(',')
    l = float(l)
    w = float(w)
    if l >= 4 and w >= 5:
        planks += 1
    elif l >= 4:
        poles += 1
    elif w >= 5:
        shingles += 1
    else:
        sticks += 1
print("There were {} planks, {} poles, {} shingles, amd {} sticks".\
         format(planks, poles, shingles, sticks))
```

8. (**12 points**) Consider the iterative merge sort algorithm.

(a) (4/12 pts) As mentioned in the practice for this exam, this algorithm can be made much faster for certain inputs if we use the concept of "runs", where a "run" in a list is a sequence of consecutive values that are non-decreasing. For example in the list

```
v = [ 7, 5, 9, 11, 2, 6, 10, 18, 19, 17 ]
```

there are four runs: one of length 1 starting at index 0, the second of length 3 starting at index 1, the third of length 5 starting at index 4, and the fourth starting at index 9 of length 1. Write a function called `make_runs` that takes a list as an argument and returns a list of runs (themselves lists), For example,

```
print( make_runs( [7, 5, 9, 11, 2, 6, 10, 18, 19, 17] ))
print( make_runs( [5, 5, 5] ))
print( make_runs( [] ))
```

should output

```
[[7], [5, 9, 11], [2, 6, 10, 18, 19], [17]]
[[5, 5, 5]]
[]
```

Solution:

```
def make_runs(L):
    if len(L) == 0:
        return(L)
    newL = []
    localL = [L[0]]
    for index in range(1, len(L)):
        if L[index] >= localL[-1]:
            localL.append(L[index])
        else:
            newL.append(localL)
            localL = [L[index]]
    newL.append(localL)
    return newL
```

(b) (6/12pts) Now modify the iterative `merge_sort()` algorithm to make use of `make_runs()` in place of the list initialization. The `merge_sort` routine is included here for your convenience.

```
def merge_sort(v):
    if len(v) <= 1:
        return v
    lists = []
    for item in v:
        lists.append([item])
    i = 0
    while i < len(lists)-1:
        new_list = merge(lists[i], lists[i+1])
        lists.append(new_list)
        i += 2
    return lists[-1] # lists[-1] contains the sorted values
```

Solution:

```
def merge_sort(v):
    if len(v) <= 1:
        return v
    lists = make_runs(v)
    i = 0
    while i < len(lists)-1:
        new_list = merge(lists[i], lists[i+1])
        lists.append(new_list)
        i += 2
    return lists[-1]
```

(c) (2/12pts) Finally, assuming your `make_runs()` function runs in O(N) time, what is the running time of the modified `merge_sort()` when the input is already sorted?

Solution:

```
O(N)
```

9. (**12 points**) Write a recursive function `depth` that takes two parameters a list `L` and an integer `d`, respectively. The function should calculate the maximum amount of nesting for the list. The parameter `d` indicates the current depth of recursion. Think back to our Sierpinski triangle example for ideas on how to use it. For example,

```
>>> print("Maximum depth: ", depth([], 0))
Maximum depth:  1
>>> print("Maximum depth: ", depth([[],[]], 0))
Maximum depth:  2
>>> print("Maximum depth: ", depth([[[1], 2],[]], 0))
Maximum depth:  3
>>> print("Maximum depth: ", depth([[],[[[]]]], 0))
Maximum depth:  4
```

Remember that you can use the `type` function to check if a variable is a `list`.

Solution:

```
def depth(L, i):
    if type(L) != list:
        return i
    D = [i+1]
    for l in L:
        D.append(depth(l, i+1))
    return max(D)
```

10. (**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, 05/09 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 05/10 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 05/11 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 05/11.

    (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 Summer!
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.