UNIVERSITY OF TORONTO

Fall 2013 Midterm #2

CSC180: Introduction to Computer Programming

Duration: 90 minutes

November 11th, 2013

Last Name: ______________________

First Name: ______________________

Student Number: ______________________

Circle your instructor: Ashraf Al Daoud Kaveh Aasaraai Shobhit Jain

Instructions:

- **Write your name on the back of the final page of this exam.**
- **Do not open this exam until you hear the signal to start.**
- Have your student ID on your desk.
- No aids permitted other than writing tools. Keep all bags and notes far from your desk before the exam begins.
- There are 5 questions on 18 pages including this page.
- Exams written with pencil will not be remarked.
- All your answers should be based on Python 3.
- Write comments where it would clarify your code. Docstrings are required only where explicitly indicated.
- If you use any space for rough work, clearly indicate the section(s) that you want to be marked.

Mark Breakdown

Q1:	/12
Q2:	/8
Q3:	/5
Q4:	/7
Q5:	/8
Q5(bonus):	/1
Total:	**/40**

Q1 (12 marks) ***(1 mark per correct print statement)***

Write the output of the execution of the following code snippets in the provided space. If the code would cause an error, write ERROR and give brief explaination. Write ***as clearly and legibly as possible***.

Code	Output

`fruits = [’apple’, ’orange’, ’banana’]`
`print(fruits[1:-1])`
`print(fruits[-2])`
`print(fruits[0:-3])`
`print(fruits[::2])`
	`[’orange’]`
	`’orange’`
	`[]`
	`[’apple’, ’banana’]`

`veggie = [’onion’, ’garlic’, ’tomato’]`
`print(len(veggie + [’potato’]))`
	`4`

`veggie = [’eggs’, ’bread’, ’butter’]`
`print(len(veggie.append(’jam’)))`
	`Error (len on None)`

`table1 = [[1, 2], [3, 4]]`
`table2 = table1[:]`
`table1.append([5, 6])`
`table1[0][1] = 1`
`print(table1)`
`print(table2)`
	`[[1,1],[3,4],[5,6]]`
	`[[1, 1], [3, 4]]`

`d = {1:’one’, 2:’two’, 3:’three’}`
`print(d[2][2])`
	`’o’`

`d = {’a’:’A’, ’b’:’B’, ’c’:’C’}`
`print(d.keys().pop())`
	`Error (dict_keys does not have pop)`

`print(type((1)) == tuple)`	`False`
`print([5, 4, 3, 1].sort())`	`None`

Q2 Short code/answers (8 marks)

Write your code in the space provided. **Do not write functions**.

Part (a) (2 marks)

Write a Python expression which evaluates to True when **exactly one** of the three boolean variables `sunny, cloudy` and `rainy` is True.

sunny + cloudy + rainy == 1

(both are correct)

(sunny and (not cloudy) and (not rainy)) or ((not sunny) and cloudy and (not rainy)) or ((not sunny) and (not cloudy) and rainy))

Part (b) (2 marks)

Reverse *IN PLACE* a list without using list.reverse() method. You should not be creating a new list.

```
for i in range(len(L) // 2):
    L[i], L[len(l) - 1 - i] = L[len(l) - 1 - i], L[i]
```

Part (c) (2 marks)

The following function is supposed to return a list of first `n` Fibonacci numbers. Fibonacci numbers are non-negative integers in the following sequence:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
```

First two numbers in the Fibonacci sequence are always `0` and `1`, and each subsequent number is the sum of the previous two. Indicate if the function has errors. If it does, suggest your corrections. You should clearly write down the line number which needs to be corrected and the corresponding correction. No marks will be awarded for just writing down the line numbers.

```
 1  def fibonacci(n):
 2      ''' (int) -> list
 3      Return a list of first n fibonacci numbers.
 4      '''
 5
 6      count = 0
 7      result = [0, 1]
 8
 9      while count < n:
10          result.append(result[count - 1] + result[count]
11          count += 1
12
13      return result
```

One possible solution (there are other solutions). line 6: count = 1
line 9: while count < n - 1:

Full mark for a working version of the code.

Part (d) (2 marks)

In any text, parentheses should be always matched such that any opening parenthesis has a corresponding closing one. For example:

- $\cdots(\cdots(\cdots)\cdots)\cdots$ # match
- $\cdots(\cdots)\cdots)\cdots$ # mis-match
- $\cdots)\cdots(\cdots)\cdots(\cdots$ # mis-match

Complete the following function using as few lines of code as possible (no mark for unnecessary complicated code).

```
def check_parentheses(file_name):
    ''' (str) -> bool

    Return True if all parentheses match (i.e., for every left
    parenthesis there exists a right parenthesis) in file file_name.
    '''

    left_count = 0 # left parentheses count
    right_count = 0 # right parentheses count

    file_object = open(file_name)
    text = my_file.read()
    file_object.close()

    for letter in text:
        if letter == '(':
            left_count += 1
        elif letter == ')':
            right_count += 1




            if right_count > left_count:
                return False

    return right_count == left_count
```

1 mark for the if statement and 1 mark for the return.

Q3 (5 marks)

We use word processors in our daily lives. One of the key features in these programs is spell checking. You are given the task of writing a spell checker for a word processing program. Your program receives the text and is supposed to return all the words that are misspelled in that text.

Write the function `spell_check(ref_words, line)` that receives two parameters: a reference list of all the words with their correct spelling, and one line of text. The function must return a list of all the misspelled words in the given line along with their indexes. The returned list must be a list of two-element lists, that is a two-dimensional list.

Your function only checks for exact matches with words in the reference word list. You do not have to check for word suffixes such as apostrophes ('s) or cases. The words in the input line are separated using space/tab characters. No punctuation (periods or commas) is present in the input line.

Example:
function parameters:

```
ref_words = ["hello", "hi", "bye"]
line = "hello HI bye goodbye hell"

return value = [["HI", 6], ["goodbye", 13], ["hell", 21]]
```

```
def spell_check(ref_words, line):

    line = line.strip().split(" ")
    index = 0
    result = []
    for word in line:
        if word == '':
            index += 1
        else:
            if word not in ref_words:
                result.append([word, index])
            index += len(word)
            index += 1
    return result
```

Important points:

1. Extracting words from the string.
2. Checking the words in the reference list.
3. building a nested list and returning.
4. returning the list

Q4 (7 marks)

You are given a file with comma separated values, representing information about various stores. Each line holds the store name and all the products that specific store sells. The first column in the file is always the store name, followed by one or more product names.

Example store file:

store.txt:

```
Walmart, diaper, razor
KFC, chicken, cholesterol
Target, tape, diaper
```

Part (a) (4 marks)

Your task is to write down the function `store_information(store_file)` which takes the store information file name as parameter, reads the content of this file into a suitable Python data structure and returns that data structure. Use a data structure of your choice which you find suitable for this task. You may find that your task in **Part b** will affect your choice of the data structure in this part.

```
def store_information(store_file):

    file_object = open(store_file)

    store_data = {}
    for line in file_object:
        line = line.strip().split(",")
        for product in line[1:]:
            if product not in store_data:
                store_data[product] = []
            store_data[product].append(line[0])

    return store_data
```

You can continue with part (a) in this space.

Important points:

1. Correctly opening and reading the contents of the file.
2. Storing the contents in a data structure.
3. returning the data structure.

Part (b) (3 marks)

In this part your task is to write down the function `find_stores(store_data, product_name)` which takes the data structure returned by `store_information` function and one product name as parameters and returns a list of all the stores that sell that specific product.

Read the given **starter code** carefully. Make sure your two functions use the same data structure for store information. If your solution to **Part a** is incorrect, you will still get marks for a correct solution to **Part b**.

```
def find_stores(store_data, product_name):

    return store_data[product_name]
```

```
if __name__ == '__main__':

    store_data = store_information('store.txt')
    store_list = find_stores(store_data, 'diaper')
    print(store_list) # Example Output: ['Walmart', 'Target']
```

Important points:

1. Using the same data structure in part (a)
2. Correctly search the product.
3. return a list

Q5 (8 marks)

A doubly stochastic matrix is a square matrix of non-negative numbers with each row and each column sum to 1. Write a function `doubly_stoch` which takes a matrix (nested list of numbers) as an input. You can assume that all the nested lists have same size and the matrix will not be empty. Your function should do the following:

1. Return `False` if the input matrix is not square.
2. Return `False` if the input matrix has at least one negative number.
3. Return `True` if the matrix is doubly stochastic (i.e. each row and each column sum to 1), and `False` otherwise.

Bonus mark: You will get 1 bonus mark for writing down an efficient solution.

```
def double_stoch(matrix):

    if len(matrix) != len(matrix[0]):
        return False

    for i in range(len(matrix)):
        sum_row = 0
        sum_col = 0
        for j in range (len(matrix)):
            if matrix[i][j] < 0:
                return False

            sum_row += matrix[i][j]
            sum_col += matrix[j][i]

        if sum_row != 1 or sum_col != 1:
            return False

    return True
```

Important points:

1. Returning False for non-square matrices.
2. Returning False for negative numbers.
3. Checking row sums.
4. Checking column sums.

Short Python function descriptions:

```
abs(number) -> number
    Return the absolute value of the argument.

len(object) -> int
    Return the number of items of a sequence or mapping.

max(iterable) -> value
max(a, b, c, ...) -> value
    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.

min(iterable) -> value
min(a, b, c, ...) -> value
    With a single iterable argument, return its smallest item.
    With two or more arguments, return the smallest argument.

open(name) -> file object
    Open a file with the given name and returns a file object.

range([start,] stop[, step]) -> list of ints
    Return a list containing an arithmetic progression of ints.
    range(i, j) returns [i, i+1, i+2, ..., j-1]; start defaults to 0.
    When step is given, it specifies the increment (or decrement).

raw_input([prompt]) -> string
    Read a string from standard input.
    The prompt string, if given, is printed without a trailing newline
        before reading.

float(x) -> float
    Convert a string or number to a floating point number, if possible
       .

int(x) -> int
    Convert a string or number to an int, if possible. A floating
       point argument
    will be truncated towards zero.

str(x) -> str
    Return a nice string representation of the object, if possible.

list(iterable) -> list
    Return a list containing the elements in iterable.
```

```
tuple(iterable) -> list
    Return a tuple containing the elements in iterable.
```

Short Python method descriptions:

```
dict:
    D.get(k[,d]) -> D[k] if k in D, else d (d defaults to None).
    D.has_key(k) -> True if D has a key k, else False.
    D.items() -> new view of D's (key, value) pairs.
    D.keys() -> new view of D's keys.
    D.pop(k[,d]) -> object -- remove specified key and return the
       corresponding value.
        If key is not found, d is returned if given, otherwise Error.
    D.values() -> new view of D's values.

file:
    F.readlines() -> list of strings, each a line from the file F.
    F.readline() -> string, single line from the file F.

list:
    L.append(object) -- append object to end of L.
    L.count(object) -> return the number of occurrences of value in L.
    L.extend(iterable) -- extend L by appending elements from the
       iterable.
    L.index(value) -> int -- return index of the first occurrence of
       value in L.
    L.insert(index, object) -- insert object into L at position index.
    L.pop([index]) -> item -- remove and return item at index in L (
       default index -1).
    L.remove(value) -- remove the first occurrence of value from L.
    L.reverse() -- reverse L (in place).
    L.sort() -- sort L in ascending order (in place).

str:
    S.count(sub) -> number of non-overlapping occurrences of sub in S.
    S.endswith(suffix) -> True if S ends with suffix, False otherwise.
        suffix can also be a tuple of strings to try.
    S.find(sub) -> lowest index in S where sub is found. Return -1 on
       failure.
    S.isalnum() -> True if all characters in S are alphanumeric and
       there is
        at least one character in S, False otherwise.
    S.isalpha() -> True if all characters in S are alphabetic and
       there is
```

```
      at least one character in S, False otherwise.
    S.isdigit() -> True if all characters in S are digits and there is
      at least one character in S, False otherwise.
    S.isspace() -> True if all characters in S are whitespace and
     there is
      at least one character in S, False otherwise.
    S.join(list) -> the concatenation of the strings in the sequence,
     with separator S.
    S.lower() -> a copy of S converted to lowercase.
    S.replace(old, new[, count]) -> a copy of S with all occurrences
     of old replaced
      by new. If count is given, only the first count occurrences
         are replaced.
    S.rfind(sub) -> the highest index in S where substring sub is
     found, or -1 if
      sub does not occur within S.
    S.rstrip([chars]) -> a copy of the string S with trailing
     whitespace removed.
      If chars is given and not None, remove characters in chars
         instead.
    S.split([sep [,maxsplit]]) ->  a list of the words in the string S
     , using sep as the
      delimiter string. If maxsplit is given, at most maxsplit
         splits are done.
      If sep is not specified or is None, any whitespace string is a
          separator and
      empty strings are removed from the result.
    S.startswith(prefix) -> True if S starts with the specified prefix
     , False otherwise.
      prefix can also be a tuple of strings to try.
    S.strip([chars]) -> a copy of S with leading and trailing
     whitespace removed.
      If chars is given and not None, remove characters in chars
         instead.
    S.upper() -> a copy of S converted to uppercase.

tuple:
    T.count(value) -> the number of occurrences of value in T.
    T.index(value, [start, [stop]]) -> the first index of value in T
```

Last Name: ______________________________

First Name: ______________________________