Student ID Number: ________________________

University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Spring 2018 Instructor: Prof. Gerald Friedland 2018-02-22

CS88 Midterm Exam

Last *Name (Please print clearly)*	Solution	
First *Name (Please print clearly)*		
Student ID Number		
What time is your lab on Monday?		
Name of the person to your: Left | Right		
All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS88 who haven't taken it yet. ***(please sign)***		

Instructions

- Don't Panic! This booklet contains 8 pages including this cover page. Put all answers on pages 2-7; you can use page 8 for extra/doodle space. Please don't hand in any stray pieces of paper.
- Please turn off all pagers, cell phones and beepers. Remove all hats and headphones.
- You have 50 minutes to complete this exam. The midterm is closed book, no computers, no PDAs, no cell phones, no calculators, but you are allowed one double-sided sheets of notes and the midterm study guide. There may be partial credit for incomplete answers; write as much of the solution as you can. When we provide a blank, please fit your answer within the space provided.
- Remember: Whatever your score in this exam – you can clobber it with the finals. If you are caught cheating, however, it's an F and you will not be able to clobber.

Good luck!

Question	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Total**
Points	**2**	**2**	**2**	**2**	**2**	**12**	**14**	**4**	**40**

Student ID Number: ________________________

Warm-up Questions with short answers (2pts each)

Please write your answer within the designated boxes. We drop the lowest-scoring question in this section.

Question 1: TRUE or FALSE: "In general, recursion is more powerful than a for loop". Explain why.

False
Iterative algorithms can be converted to recursive and vice-versa. See also question 8.

Question 2: TRUE or FALSE: "A function does not allow repeated execution of statements". Explain why.

False
For loops, while loops, recursion etc. are some tools that can be used to allow for repeated execution of statements within a function definition.

Question 3: Look at the box below. There is a list of problems on the left (labeled 1–4) and a list of the concepts that solve these problems (labeled A–D). **Write** the correct letter to the left of each number to match each problem to the Internet component that you could use to solve that problem.

__B___	1. Step through a list	A. Operator
__A___	2. Manipulate a Literal	B. Iterator
__D___	3. Shield a user from a concrete implementation	C. Higher Order Functions
__C___	4. Create a function that returns a function	D. Abstraction

Question 4: When analyzing an algorithm's running time, we count the number of operations or "frames" instead of timing it with a stopwatch. List one benefit and one disadvantage of this decision.

Benefits:
Hardware independent, focuses on asymptotic runtime (i.e. big picture), stop watch isn't easily reproducible, It is easy to count number of operations, allows for idea of memory usage.

Disadvantages:
Gives you a theoretical speed instead of the actual speed, need to understand code to count the number of operations while this isn't true for using a stopwatch, some frames execute at different runtimes than others

Question 5: Provide two practical reasons why Python allows to define a function within a function

1. Allows for more complex functionality
2. Abstraction / Makes the code shorter and more organized
3. Saves space in the global frame and reduces clutter if the inner function is only required in the scope of the outer function
4. Since functions are a data type, they can be used as input values or returned as outputs

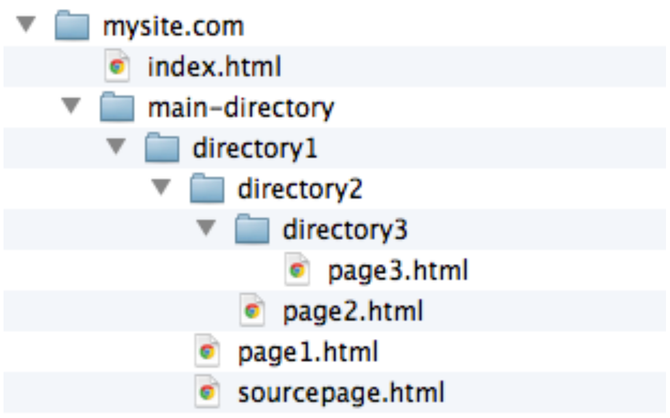
Student ID Number: ________________________

Question 6: ***File Search…*** (12 pts)

In today's computers, directories (also called folders) and files are used to structure information on hard disks and memory cards. Directories contain files and also other directories. Directories can also be empty. The uppermost directory is called /.

Write a function **`searchFile(name, dir)`** that searches for the file **`name`** in an arbitrary large and arbitrary shaped directory structure starting in directory **`dir`**, and reports **`true`** if the file exists or **`false`** if it doesn't. Note that it is possible for a file and a directory to have the same name.

Example of a Simple Directory Structure:



For example in the above picture:

> **`searchFile(page1.html, mysite.com)`** should return **`true`**.
> **`searchFile(page4.html, mysite.com)`** should return **`false`**.

Here are two helper functions to help create your solution:

1. **`listContent(dir)`** – gives you all the files and directories in a directory as a list of names (as Strings) in the specified directory (no recursion)
2. **`isDir(name)`** – reports **`true`** if **`name`** is a directory and **`false`** if it is a file

(6a) Let's start by creating a helper function **`getFiles(dir)`** that takes in a directory and reports a list of just the files in that directory. The solution should fit in one line.

```
def getFiles(dir):

    return ( [f for f in listContent(dir) if isDir(f)==false] )
```

Student ID Number: ________________________

(6b) Create another helper function **getDir(dir)** that takes in a directory and returns a list of sub-directories, and [] if none exist. You may not need all the lines.

```
def getDir(dir):

        return ( [d for d in listContent(dir) if isDir(f)==true] )

    ____________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________

    ______________________________________________________________________
```

(6c) Now assume your helper functions **getFiles(dir)** and **getDir(dir)** work correctly. You may use them along with any other blocks to create the **searchFile(name, dir)** solution.

```
def searchFile(name, dir):
     if name in getFiles(dir):
         return true
     else:
         i = False
         for d in getDirs(dir):
            i = i or searchFile(name,d)
         return i
```

Student ID Number: ________________________

Question 7: *Checksum…* (14 pts)

A checksum is a function that calculates a particular value for a set of characters in order to make sure there are no errors. *Luhn's Algorithm* is a way of verifying whether a credit card number is valid. It does this by calculating a *checksum* of the card number that is only correct when all digits are also correct. Checksum algorithms are used every time you use the internet, to make sure the data being sent is transferred correctly.

Let's define a very simple checksum algorithm: *Count every 'a' as 1, 'b' as 2, …, 'z' as 26, sum the values together. For example, the string "Cab" would be 3(C) + 1(a) + 2(b) = 6 points.*

We will give you two helper functions:

1. **`letterTovalue(letter)`**, which if given **`a`** or **`A`** returns 1, **`b`** or **`B`** returns 2, etc.
2. **`wordTolist(word)`**, which if given the word **`Cab`**, it would return the list **`['C','a','b']`**

You can also use higher-order functions and arithmetic operations as usual.

(7a) Write **`checksum(word)`** that returns the checksum of the word. E.g., **`checksum(Cab)`**➔**6**

```
def checksum(word):

    total = 0
    l = wordTolist(word)
    for item in l:
        total += letterTovalue(item)
    return total
    ____________________________________________________________
```

(7b) Give an example where this checksum algorithm would fail (i.e. where two different strings will give the same checksum value) and explain why.

Example 1 (two strings):

Why it fails:

Since addition is commutative, any word with the same letters would get the same checksum.

Student ID Number: ________________________

(7c) Now assume that there are only 9 letters in the alphabet (a – i), each with the same “letter-value” as before (*‘a’ as 1, ‘b’ as 2, …, ‘i’ as 9*). Also assume that a given word cannot have any duplicate letters. Implement an improved algorithm in which every valid input reports a unique output. Explain verbally how your algorithm works, and how it fixes the problem from our original checksum.

In addition, you may use these two functions useful:

1. **pow(x,y)** – takes in numbers **x** and **y**, and reports **x** to the power of **y**
2. **pos(word, letter)** – takes in a letter and a word, and reports the position of the letter in the word

```
def uniqueChecksum(word):

    total = 0
    l = wordTolist(word)
    for item in l:
         total += letterTovalue(item) * pow(10, pos(word, item))
    return total
```

How does your algorithm work, and how does it fix the problem from our original checksum?

This algorithm works because it takes both the position of the letter in the word and the value of the letter into account. Additionally, multiplying the letter value by 10^{pos} ensures that all cases are accounted for.

Note: Simply multiplying the position of the letter by the letter value doesn’t work in every case (e.g. uniqueChecksum(“abc”) -> (1*1 + 2*2 + 3*3 = 14) and uniqueChecksum(“n) -> (1*14) would be the same.

Student ID Number: ______________________

Question 8: ***Recursion to Iteration*** (4 pts)

Rewrite the following function as an iterative function (Note you cannot use ** or the pow function):

```
def power(x, n)
    if n = = 0:
           return 1
    else
           return x * power(x, n-1)


def power_iter(x, n):
    result = 1
    i = 0
    while i < n:
       result = result * x
       i += 1
    return result
```

Student ID Number: ________________________

Doodle/Notes/Extra Space: