

Student ID Number: 3033 285 244

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

<i>Last Name (Please print clearly)</i>	Braude
<i>First Name (Please print clearly)</i>	Omer
<i>Student ID Number</i>	3033 285 244
<i>What time is your lab on Monday?</i>	4:00-6:00
<i>Name of the person to your: Left   Right</i>	Alternative
<small>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)</small>	lm

### 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 sheet 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.

There are some mistakes!

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: 3033285244

### 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, everything can also be done with a for loop. Sometimes it is simpler and more readable to use recursion.

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

True. For or while loops are repeated execution of statements and that what makes programming powerful.

**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.

<u>B</u> 1. Step through a list	A. Operator
<u>A</u> 2. Manipulate a Literal	<input checked="" type="checkbox"/> B. Iterator
<u>D</u> 3. Shield a user from a concrete implementation	<input checked="" type="checkbox"/> C. Higher Order Functions
<u>C</u> 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.

Benefit: We get the same result independent of the computer we use.  
Disadvantage: Sometimes we need a more useful information which is actually the time that the operation will take

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

1) It makes the code easier to read because we will know that this function is used only in the other function and not relevant for the rest of the code. 2) It makes it easier to collaborate with other people, since one person can work on the first function and deliver it as is (basically the idea of abstraction).

Student ID Number: 3033285244

### 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 ( [x for x in listContent(dir) if !isDir(x)] )
```

Student ID Number: 3033285244

(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 [x for x in listContent(dir) if isDir(x)]
```

(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 getDir(dir) == [] and name not in getFiles(dir):
```

```
        return False
```

```
    elif name in getFiles(dir):
```

```
        return True
```

```
    else:
```

```
        for i in getDir(dir):
```

```
            if searchFile(name, i):
```

```
                return True
```

```
    return False
```

Student ID Number: 3033285244

**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):  
    sum = 0  
    for i in wordToList(word):  
        sum = sum + letterToValue(i)  
    return sum
```

(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):

'cab'	'bbb'
-------	-------

Why it fails:

the function I made will sum up 3+1+2 for the first string and 2+2+2 for the second one. So both give value of 6.

Student ID Number: 3033 285 244

(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

I assume that for the first letter `pos` return position 0.

```
def uniqueChecksum(word):
```

```
    sum = 0
```

```
    for i in wordToIndex(word):
```

```
        sum = sum + pow(letterToValue(i), 3 * pos(word, i) + 1)
```

```
    return sum
```

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

The algorithm makes a sum of the letters but each letter is raised to the power of 3 times its position. This difference makes the position of the letter crucial and spread the values of sum over a large range (I multiply by 3 to make the space even larger just in case) I am certain that each combination will now return a unique value.

Student ID Number: 3033 285244

**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):
```

```
    if n == 0:
```

```
        return 1
```

```
    elif n == 1:
```

```
        return x
```

```
    else:
```

```
        result = 1
```

```
        for i in range(n):
```

```
            result = result * x
```

```
        return result
```

Student ID Number: \_\_\_\_\_

Doodle/Notes/Extra Space:

