University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2018 Instructor: Prof. Gerald Friedland Real Final: 2018-05-09

Computational Structures for Data Science, CS88 **Mock** Final Exam

Last Name (Please print clearly)	**Solutions**	
First Name (Please print clearly)		
Student ID Number	**Do not turn this in.**	
What time is your lab?		
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)**	We advise you take the mock final by yourself a couple days before the final and then discuss results with your peers.	

Instructions

- Don't Panic! This booklet contains 11 pages including this cover page. Put all answers on pages 2-11; you can use page 12 for extra/doodle space.
- Please turn off all pagers, cell phones and beepers. Remove all hats and headphones.
- You have 90 minutes to complete this mock exam (the actual final will be two hours). The final is closed book, no computers, no PDAs, no cell phones, no calculators, but you are allowed two double-sided sheets of notes, the midterm study guide and the final 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 – it counts 20% of the total grade. If you are caught cheating, however, it's an F and we will have to report it.
- You are allowed to use standard Python data structures for programming questions.

Good luck!

Question	1	2	3	4	5	6	Total
Points	5	6	5	6	6	12	40

Warm-up Questions with short answers (1pt each)

Please write your answer within the designated boxes.

Question 1a: What is Polymorphism in object-oriented programming?

Polymorphism refers to the programming language's ability to implement methods with the same name within different classes in the inheritance tree differently (Lecture 11)

Question 1b: Give one example which will match the following pattern: r'^[^aeiou]\d.+'

Any string which begins with a character that is not a vowel and has either 1-9 as the second character.
Example: 'B2222', 'C2', '123', 's2hoes'

Question 1c: Fill in the gaps with some of the following words so that the sentences are correct: class, instance, object, class variables, specialized, inherits, overloads, instance variables, methods.

A class consists of attributes and __methods___that define its behavior. __Class variables___ are attributes shared among all objects of a class, while _instance variables____ are attributes specific to an object. A child class __inherits___ attributes from a superclass or parent class.

Question 1d: How does Python handle errors?

[Lecture 8] Python handles errors by terminating immediately and printing an error message.

Question 1e: Give two benefits of object-oriented programming.

- Design benefits: Makes it easier to write very large programs / applications
- Code reuse: Inheritance saves the amount of code that has to be written

Question 2: SQL (6 pts)

Use the two tables below, hotels and guests, for the questions in this section.

hotels

hotel_id	name	location	avg_ratings
1	Tipton Hotel	San Francisco	4.7
2	Feraton Hotel	Maui	4.6
3	Hotel Tarriot	Los Angeles	4.6
4	TripleTree Hotel	San Diego	4.3

guests

name	hotel_id	rating_given
Andrea	3	4.8
Justin	4	4.0
Eric	1	4.7
Sherry	1	5.0
Lauren	2	4.9
Spencer	3	4.4
Joyce	4	4.3

cities

name	top_attraction
San Francisco	Golden Gate Bridge
Maui	Haleakala Volcano
Los Angeles	Disney Land
San Diego	San Diego Zoo

1. (2 points) Fill in the blanks for the SQL query below. The query returns a table with 2 columns: hotel_name and the number of guests who stayed there. The column names don't matter.

```
SELECT h.name, count(g.name)
FROM hotels as h, guests as g
WHERE h.hotel_id = g.hotel_id
GROUP BY g.hotel_id;
```

2. (2 points) What does the following query return? Write down all output values AND column names. You may not need all the lines.

```
SELECT name
FROM guests
WHERE rating_given > 4.5;
```

Name
Andrea
Eric
Sherry
Lauren

3. (2 points) Write a query to return one column with the names of the guests who are staying in the hotel which is located in the city with the volcano as the top attraction .

```
SELECT g.name
FROM GUESTS as g, HOTELS as h, CITIES as c
WHERE c.top_attraction = "Haleakala Volcano" AND c.name = h.location AND
h.hotel_id = g.hotel_id;
```

Question 3: *Sequences and Generators* (5 pts)

1. Write a generator that takes in any number of iterables and zips them together. It should output a series of lists, each containing the nth items of each iterable. It should stop when the smallest iterable runs out of elements.

```
def zip_generator(*iterables):
      """
      >>> z = zip_generator([1, 2, 3], [4, 5, 6], [7])
      >>> for i in z:
      ...     print(i)
      ...
      [1, 4, 7]
      >>> z = zip_generator([1, 2, 3], [4, 5, 6], [7, 8, 9, 10])
      >>> for i in z:
      ...     print(i)
      ...
      [1, 4, 7]
      [2, 5, 8]
      [3, 6, 9]
      """
      iterators = [iter(i) for i in iterables] # create a list of
      iterators
      while True:
            # yield the next element from each iterator in “iterators”
            yield [i.__next__() for i in iterators]
```

Question 4: *Buggy Code* (6 pts)

Each of the functions below has an error that causes the function to have different behavior than what is specified in the docstring. The output of the error is given to you. Use the code and error output to explain the bug in 2 sentences or less. Violation of an assert statement does not count as a valid bug. For example, calling countdown with a negative number should error due to the assert statement. This is not a bug.

1.
```
def convert_lists_to_dictionary(list1, list2):
‘‘‘
>>> convert_lists_to_dictionary ([a, b, c], [1, 2, 3])
{‘a’: 1, ‘b’: 2, ‘c’:3}
’’’
        assert type(list1) == list
        assert type(list2) == list
        assert len(list1) == len(list2)

        dict = {}
        dict[list1] = list2 => dict[ [a,b,c] ]
Dict[“a”] = 4
```

Error message:
TypeError: unhashable type: 'list'

Explanation:

The type dictionary keys has to be immutable hence you cannot use a list as a dictionary key.

2.
```
def list_sum(list):
        ‘‘‘
        >>> list_sum([1,2,3])
        6
’’’
```

```
        if len(list) == 1:
            return list[0]
    else:
            return list[0]+list_sum(list[0:])
```

Error message:
RuntimeError: maximum recursion depth exceeded while calling a Python object

Explanation:

The recursive case does not reduce the length of the list (since list[0:] returns the same list) hence it never moves towards the base case and will run for ever.

Question 5: *Trees* (6 pts)

Use the following implementation of the binary tree class for the questions below. The entry in each node of the tree must be an integer.

```
class Tree:
        def __init__(self, entry, branches=()):
assert type(self.entry) == int
        self.entry = entry
for branch in branches:
        assert isinstance(branch, Tree)
        self.branches = list(branches)

def is_leaf(self):
        return not self.branches
```

Write a method for the Tree class that will return the minimum value of all leaf nodes in a Tree object.

```
def tree_leaf_min(self):
        '''
        >>> t = Tree(2,[Tree(3),Tree(5,[Tree(10)])]) >>>
        t.tree_leaf_min()
        3
        '''
        if self.is_leaf():
                return self.entry
        else:
                return min([b.tree_leaf_min() for b in self.branches])
```

Question 6: *WWPP* (12 pts, 2 each)

Consider the following class definition.

```
class pet(object):
        def __init__(self,weight):
                self.weight = weight
        def getweight(self):
                return self.weight
        def about(self):
                return "This is a pet"

class insurable(object):
        def __init__(self,amount,age):
                self.amount = amount
                self.age = age
        def getinsurancecost(self):
                return self.amount * self.age
        def about(self):
                return "This is an insurable"

# class dog inherits from pet object and insurable object
class dog(pet, insurable):
        def __init__(self,sound,weight,amount, age):
                self.sound = sound
                pet.__init__(self,weight)
            insurable.__init__(self,amount,age)

        def getnoise(self):
                return self.sound

        def about_dog(self):
                rv = self.about() + " " + self.getnoise() + " " + \
                str(self.getweight()) + " " + \
                  str(self.getinsurancecost())
               return rv
```

hound = dog("woof",31,1000,3)
parrot = pet(8)

For each of the following, fill in what Python would print. Some might take more than a line, some might not need every line. If there is an error, specify what kind of error. If Python wouldn't print anything, leave the lines blank.

>>> print(hound.getnoise())
>>> print(hound.getweight())

"woof"
31

>>> print(hound.about_dog())

"This is a pet woof 31 3000"

>>print (hound.about())

"This is a pet"

>>> print(parrot.getnoise())

AttributeError

>>>print (parrot.getinsurancecost())

AttributeError

>>>print (hound.getinsurancecost())

3000

Question 7: *Dictionaries*

There's a new word-guessing game going around that everyone is playing called Max Vowels. As the name implies, you receive more points if you guess a word with a larger amount of vowels. Within a given time limit, you guess as many words as you can and you calculate the amount of points you have by going through each word you guessed and adding up all the vowels in each of these words.

But you realize it's really slow to count the number of vowels for each word every time you start the game over. To make counting the number of points for each player faster, you instead choose to categorize each word by the number of vowels there are in that word. So for example, the number 3 would correspond to 'elephant', 'capsize', and 'patio'. This way, you can just check how many words there are for each category and multiply that by the category to find the total number of points you have.

Write the following function that will return a dictionary that will store each word according to the number of vowels there are in the word.

```
def store_words(words):
    """
    >>> guesses = store_words(['i', 'love', 'CS88'])
    >>> guesses
    {0: ['CS88'], 1: ['i'], 2: ['love']}
    >>> guesses[1]
    ['i']
    >>> 3 in guesses
    False
    """
    stored_words = {}

    def num_vowels(word):
        num_vowels = 0
        vowels = ['a', 'e', 'i', 'o', 'u']
        for letter in word:
            if letter in vowels:
                num_vowels += 1
        return num_vowels
```

```
    for word in words:
        category = num_vowels(word)
        if category not in stored_words:
            stored_words[category] = [word]
        else:
            stored_words[category] += [word]
    return stored_words
```

Now, count the total number of points there are in the player's dictionary of guessed words.

```
def count_points(words):
    """
    >>> guesses = store_words(['i', 'love', 'CS88'])
    >>> count_points(guesses)
    >>> 3
    """
    total = 0
    for category in words:
        total += category * len(words[category])
    return total
```

Doodle/Notes/Extra Space: