Practice Final Exam

Classes/Objects

1. Define a class called IceCream to help Baskin-Robbins manage their ice cream orders. An IceCream object represents a cone with varying flavors and number of scoops. An IceCream object starts out with no scoop but more can be added.

   The IceCream class should have the following methods.

Method	Description
`__init__(self)`	Creates an IceCream and sets up any fields necessary.
`add_scoop(self, flavor, num_of_scoops)`	Adds the corresponding flavor and number of scoops to IceCream.
`get_flavor(self, flavor)`	Returns the number of scoops of the given flavor in this IceCream. Returns zero if the flavor is not on the cone.
`to_string(self)`	Returns a string representation of the IceCream in the form: “<total number of scoops> scoops of ice cream with <list of flavors>”

2. You are given the following class:

```
class Recipe:

    def __init__(self, name):
        """
        Creates a new recipe with the given name, but no
        ingredients.
        """
        self.name = name
        self.ingredients = {}

    def add_ingredient(self, ingredient, amount):
        """
        Adds the given ingredient to our Recipe with the given
        amount in ounces. Only adds the ingredient if it is not
     water.
        """
        if not ingredient == "water":
            value = self.ingredients.setdefault(ingredient, 0)
            self.ingredients[ingredient] = value + amount
```

1. Fill in the following method that will be added inside the Recipe class.

```
    def get_ingredients(self, num_servings):
        """
        Return a new dictionary that contains the same
        ingredients in our Recipe but multiplies the amounts
        of each ingredient with the given num_servings

        Ex:
        self.ingredients =
        {"rice": 2.5, "banana": 13.0, "chili": 4.0}
        get_ingredients(4) returns
        {"rice": 10.0, "banana": 52.0, "chili": 16.0}
        """
```

2. Assume support for instructional steps is added, and the following method is made available inside the class:

```
def add_next_step(self, next_step):
    """Adds the provided next_step string to the object's
       existing series of steps."""
    <code>
```

Write **client** code to create a recipe object to prepare cereal. The object should be assigned to the variable `breakfast`, have the name `"Cereal"`, have the steps `"Pour the cereal"` and `"Pour the milk"` in that order, and have the ingredients `{"milk": 1, "cereal": 2}`.

Functions

3. Write a function that takes a string and returns the same string but with even indices uppercased and odd indices lowercased. You can use the upper() and lower() function for strings.
   Ex: swap_casing("hello world") should return "HeLlO WoRlD"

4. Write a function that takes in a dictionary mapping integers to strings. If the key is divisible by two, set the value equal to “even”. Return a list containing the original values of the even keys.
   Ex: even_key({2:”hello”, 3:”one”, 4: “cat”}) should return [“hello”, “cat”] and the dictionary should look like this {2:”even”, 3:”one”, 4: “even”}

Debugging

5. You receive the following error messages after running your code:

```
Traceback (most recent call last):

File "social_network.py", line 338, in <module>

do_stuff(friends_list)

File "social_network.py", line 200, in do_stuff

result = recommend_by_influence(friendlist)

File "social_network.py", line 107, in
recommend_by_influence

output = read_result()

NameError: global name 'read_result' is not defined
```

   a. List the names of the stack frames that existed at the point that the error was discovered?

   b. What is the most recent stack frame (eg. the last function that was successfully called)?

   c. Describe how you would go about trying to find the cause of and fix the error:

Function Output

6.

Write the output of the code below:

```
sum = 0
for x in range(1, 25, 2):
      temp = (x / 10) % 10
      sum = sum + temp
print("sum:", sum)
```

Documentation

7.

Write a docstring for the following function. Document the inputs, outputs, and any side effects of the function.

```
def unknown_func(num_list):
    """




    """
    unique_nums = set(num_list)
    print("There are", len(unique_nums), "unique numbers in the
list")
    factors = {}
    for num in unique_nums:
     factors[num] = set()
     for factor in range(1, num):
            if num % factor == 0:
                factors[num].add(factor)
    return factors
```

List Comprehensions

8.

These two questions refer to this list:

```
lst = [[1, 2, 3], [4, 12, 76, 12], [5, 2, 7, 1, 1]]
```

a. Write a list comprehension expression that will produce a list containing only lists in lst that have an odd length. For each of those lists, they should be sorted in reverse-numeric order. Be sure your code is not altering the original list. The output should be:

```
[[3, 2, 1], [7, 5, 2, 1, 1]]
```

b. Write a comprehension expression that will produce a dictionary whose keys are the sums of the lists from lst, and whose values are the lists themselves. You may use the sum function for this question. The output should be:

```
{6: [1, 2, 3], 104: [4, 12, 76, 12], 16: [5, 2, 7, 1, 1]}
```

9.

You are given the following class:

```
class Dog:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name
```

In your main function, you have a list of dog names:

```
dog_names = ["Spot", "Woofy", "Fluffy", ""]
```

Write a list comprehension to create a list of Dog objects, one with each name in the list of dog_names, excluding names that are empty strings.