Compsci 101: Test 1 Practice 2

September 30, 2013

Name: ____

NetID/Login: ______________

Community Standard Acknowledgment (signature) ____

	value	grade
Problem 1	12 pts.	
Problem 2	12 pts.	
Problem 3	12 pts.	
Problem 4	12 pts.	

In writing code you do not need to worry about specifying the proper `import statements`. You do not need to worry about getting function or method names exactly right. Assume that all libraries and packages we have discussed are imported in any code you write.

PROBLEM 1 : (***It's a mystery (12 points)***)

Consider the following `mystery` function with one parameter `animals` which is a list of strings.

```
def mystery(animals):                              # line 1
    ''' animals is a list of strings '''
    x = [ ]                                        # line 2
    for w in animals:                              # line 3
        x.append(len(w))                           # line 4
    amount = max(x)                                # line 5
    y = [w for w in animals if len(w) == amount] # line 6
    return y[0]                                    # line 7
```

A. (4 pts) Consider making the call `mystery(animals)` with the value of `animals` below. Answer the following questions about tracing what happens with this call

```
animals = ['cat', 'mouse', 'snake', 'chicken', 'fish']
```

A1. What is the value of `x` after line 5 executes?

A2. What is the value of `amount` after line 5 executes?

A3. What is the value of `y` after line 6 executes?

A4. What value is returned from the call `mystery(animals)`?

B. (8 pts) Consider making the call `mystery(zoo)` with the value of `zoo` below. Answer the following questions about tracing what happens with this call

```
zoo = ['lion', 'rhino','bear','zebra']
```

B1. What value is returned from the call `mystery(zoo)`?

B2. Explain in words what `mystery` does.

B3. Rewrite lines 2-5 as one line that includes a list comprehension

B4. In the original code, if line 7 was changed to `return y[-1]`, explain in words what `mystery` would now do.

PROBLEM 2 : (***Dinner Functions***)

Ellen and Oscar want an easy way to decide who will cook dinner each night. They decided that Oscar will cook if it is an odd day and Ellen will cook if it is an even day. But then Oscar realized that many times he will cook two days in a row, on the 31st and the 1st, but that Ellen would never cook two days in a row. They then agreed in addition that if the day was the 31st day of the month then if the month was even, Ellen would cook and if the month was odd, Oscar would cook that day.

Write the function `whoseNightToCook` that has two int parameters `day` and `month` and returns the string 'Ellen' or 'Oscar', the name of the person who should cook dinner that night following the rules above. Assume the arguments are correct. That is, you do not need to know or verify how many days in a month.

call	**returns**	**comment**
whoseNightToCook(13, 4)	'Oscar'	day is odd
whoseNightToCook(16, 3)	'Ellen'	day is even
whoseNightToCook(31, 8)	'Ellen'	day is 31, month is even
whoseNightToCook(31, 3)	'Oscar'	day is 31, month is odd

```
def whoseNightToCook(day, month):
    '''
    returns "Ellen" if an even day, "Oscar" if an odd day
    except when day is 31, returns "Ellen" if month is even,
        and "Oscar" if month is odd
    '''
```

PROBLEM 3 : (***The Vicissitudes of Life (16 points)***)

In some competitions such as gymnastics and figure skating several judges score a competitor's effort. The score assigned is based on the average of all the judge's scores after removing the high and low score from those from which the average is calculated.

Two functions are shown below for calculating scores. They both calculate and return the average judge-score after removing the high and low score.

For example, if the list `[5.0, 5.5, 4.5, 5.0, 5.5]` is the value of the variable `scores` then the expression `computeScore(scores)` should evaluate to 5.1667 The average of 5.0, 5.5, and 5.0 is 5.1667—note that one high score of 5.5 and the low score of 4.5 do not contribute to the average.

You're given two implementations of `computeScore` and asked to comment about features they have. Given identical lists, each implemenation returns the same results, i.e., the only differences in the functions are in the style/code, not in whether the functions are correct. *The implementations are on the next page.*

Part A (4 points)

Explain, briefly, why both functions generate an error message `ZeroDivisionError` when passed a list of two elements.

Part B (4 points)

Briefly, why is the list sorted in Implementation I?

(continued after code)

Implementation I

```
def computeScore(scores):
    scores.sort()
    return sum(scores[1:len(scores)-1])/(len(scores)-2)
```

Implementation II

```
def computeScore(scores):
    tot = sum(scores)
    low = min(scores)
    high = max(scores)
    return (tot - low - high)/(len(scores)-2)
```

Part C (8 points)

You are to write a new implementation of `computeScore` in which *all scores* equal to the high score and *all scores* equal to the low score are thrown out, i.e., they do not contribute to the average which is then multiplied by the difficulty factor. For example, for the scores `[5, 3.5, 4, 3, 4, 5, 5, 2, 3.5, 5, 2]` the average would be $(3.5 + 4 + 3 + 4 + 3.5) = 18/5 = 3.6$ since each score of 5 (the high score) and each score of 2 (the low score) do not contribute to the final average.

Complete the function below.

```
def computeScore(scores):
```

PROBLEM 4 : (***X-country Scoring (15 points)***)

In cross country running a team's score is based on where its runners place, i.e., first, second, third, . . . last. It's the place that's used, not the runner's time (but the a runner's time can be used to break ties, we won't worry about that in this problem). A team's score is calculated by adding the places of its first five finishers, lowest score wins (see examples below).

For example, consider running data stored in the format shown below in a file `"xcountry.dat"`, where each line stores the time a runner crossed the finish line, and the school of the runner, separated by a comma. The first line of the file is for the first-place finisher, the second line of the file for the second-place finisher, and so on so that in general the n^{th} line of the file is for the n^{th} finisher. In the example below a UNC runner finishes first, but Duke runners finish second, third, fifth, and ninth.

```
16:58,UNC
17:52,Duke
17:57,Duke
18:03,Wake Forest
18:07,Duke
18:10,Wake Forest
18:12,UNC
18:25,William and Mary
18:27,Duke
18:37,William and Mary
18:39,Wake Forest
18:45,Wake Forest
18:59,UNC
19:01,UNC
19:01,Duke
19:02,William and Mary
19:05,William and Mary
19:15,Duke
```

Complete the function `getScore` whose input parameters specify a file of xcountry-running data and a school and that returns the total score for the team. See the sample output below.

For example, if the function `getScore` is completed as required the output of this Python code run on the data file above is shown.

Run this code:

```
teams = ["UNC", "Duke", "Wake Forest", "William and Mary", "Georgia Tech"]
for t in teams:
    s = getScore("xcountry.txt",t)
    print t,s
```

Output follows:

```
UNC 35
Duke 52
Wake Forest 33
William and Mary 51
Georgia Tech NO SCORE
```

Note that Duke runners place 2,3,5,9, 15 and $2 + 3 + 5 + 9 + 15 = 34$. A Duke runner places 18th and in the code you write you'll add the places of all runners, not just the first five runners for a team.

Complete `getScore` on the next page, but score all runners, not just the first five. When your program runs it should report 52 for Duke because $34 + 18 = 52$.

Part A (10 points)

```
def getScore(filename,uname):
    file = open(filename)
    score = 0
```

```
    file.close()
    return score
```

Part B (5 points)

You should write an explanation of how you'd limit the code you write to score just the top five runners on a team. You don't need to write code, you can write words to indicate how you'd limit the scoring to five runners. You can write code, or you can write an explanation in English.