Test 1: Compsci 06

Owen Astrachan

October 6, 2010

Name: _____

NetID/Login: _____

Honor code acknowledgment (signature) _____

	value	grade
Problem 1	12 pts.	
Problem 2	24 pts.	
Problem 3	16 pts.	
Problem 4	15 pts.	
TOTAL:	67 pts.	

This test has 13 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes. The last page is blank, if you use it make a note for the problem.

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

PROBLEM 1 : (*Outside (12 points)*)

The Python code below generates six lines of output. Write each line of output after the print statement that generates it.

```
num = 13
d = 7.5
s = "please"

print num/2




print s[1:]




print s[3]




print num % 5


print s*2


print (num-11)**3
```

PROBLEM 2 : (***Tis a Gift to be Simple (24 points)***)

Part A

Write the function `sphereV` that returns the volume of a sphere whose radius is given by the parameter `r`. The formula for the volume of a sphere is

$$\frac{4}{3} \times \pi \times r^3$$

You can use either 3.1415 or `math.pi` for π.

call	return value
`sphereV(1)`	4.1887902
`sphereV(2)`	33.510321

```
def sphereV(r):
    """
    returns volume of sphere whose radius is given by float r
    """
```

Part B

Recall that the Python expression `"at"*3` evaluates to the string `"atatat"` and the expression `"++"*4` is the String `"++++++++"`.

Write the function `pyramid` that prints a pyramid whose number of levels is given by parameter `levels`. The function prints a pyramid, it doesn't return any value. The figure on the left below is generated by `pyramid(5)` and the figure on the right is generated by by `pyramid(3)`.

```
*                                   *
**                                  **
***                                 ***
****
*****
```

In general there are n asterisks on level n.

```
def pyramid(levels):
    """
    print a pyramid with the number of levels specified
    """
```

Part C

Write the function `lastFirst` described below that returns a String in the format `first last` given a String in the format `last, first` as shown in the examples.

call	**return value**
`lastFirst("Smith, John")`	`"John Smith"`
`lastFirst("Van Doren, Mamie")`	`"Mamie Van Doren"`
`lastFirst("Begley Jr., Ed")`	`"Ed Begley Jr."`

The String parameter `name` passed to `lastFirst` will alway contain a String in the format *"last, first"* where the only comma in `name` comes after the last name, the comma is followed by one space, and then the first name.

```
def lastFirst(name):
    """
    returns "first last" given String name in format "last, first"
    """
```

Part D

A prime number has only 1 and the number itself as divisors. For example, 11 is prime since the only divisors are 1 and 11. The number 49 is not prime since it is divisible by 7. One simple way to determine if a number is prime is to test all its possible divisors other than 1 and itself. If none of the possible divisors evenly divides the number it is prime. For example, none of the values $2, 3, 4, \ldots 12$ evenly divides 13, so 13 is prime. It's possible to test fewer potential divisors, but testing from 2 to one less than the number is fine in this problem.

Complete `isPrime` so that it returns `True` if its parameter is prime, and `False` otherwise.

call	**return value**
`isPrime(3)`	`True`
`isPrime(31)`	`True`
`isPrime(33)`	`False`
`isPrime(81)`	`False`

```
def isPrime(n):
    """
    returns True if n is prime, and False otherwise
    """
```

Part E

Complete the function `domainCount` that returns the number of email addresses in its list parameter that end with a specific suffix. All the email addresses in `adds` are properly formatted and end in either .com, .net, .edu, or .org. The parameter `suffix` is one of these email-address suffixes.

For example, if `ms` is the list shown:

```
ms =
["ola@duke.edu","fgs@gmail.com","foo@yahoo.com","mrs@yale.edu","kyj@gmail.com","sms@foob.com"]
```

call	return value
`domainCount(ms,".edu")`	2
`domainCount(ms,".com")`	4

In writing `domainCount` you may want to use either: the string method `endswith` that returns `True` if a string ends with a specific string, e.g., `"monkey".endswith("key")` evaluates to `True`; or the string method `rfind` that returns the first index from the right of a specific string, e.g., `"colorful.rfind("o")` evaluates to 3.

```
def domainCount(addrs, suffix)
    """
    addrs is a list of email addresses, properly formatted
    suffix is a valid ending for an email address
    returns number of strings in addrs that end with suffix
    """
```

Part F

Write the function `makeAcronym` that creates an acronym from the first letter of each string in its list parameter. For example:

call	return value
`makeAcronym(["self","contained","underwater", "breathing","apparatus"])`	`"scuba"`
`makeAcronym(["port","out", "starboard", "home"])`	`"posh"`

```
def makeAcronym(words):
    """
    return acronym formed by concatenating first letter
    of each string in words, a list of strings
    """

    acro = ""
```

return acro

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 18^{th} 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.

(nothing on this page)