15-112 Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue

CMU 15-112 Fall 2018: Fundamentals of Programming and Computer Science
Homework 3 (Due Sunday 16-Sep, at 5pm)

- Reminder: all problems that are not explicitly marked COLLABORATIVE must be completed individually, as stated in the course syllabus.

- To start:
  1. Create a folder named 'week3'
  2. Download hw3.py, cs112_f18_week3_linter.py, and hw3_files.zip to that folder. Unzip hw3_files.zip in the folder.
  3. Edit hw3.py using Pyzo
  4. When you are ready, submit hw3.py to Autolab. For this hw, you may submit up to 8 times, but only your last submission counts.
- Do not use lists, dictionaries, or recursion this week.
- Do not hardcode the test cases in your solutions.

---

0. Style [0 pts]
   Starting with this assignment, we will be grading your code based on whether it follows the 15-112 style guide. We may deduct up to 10 points from your overall grade for style errors. Writing your code with good style from the beginning is the best way to avoid losing points!

1. Reasoning over Code [5 pts]
   Given the function roc (shown below), find a value that will make roc return True. You should then modify the function rocAnswer() to return that value so that the autograder will accept your work.

```
import string
def roc(s):
    assert(type(s) == str)
    a = 0
    b = 0
    for i in range(1, len(s), 2):
        if s[i] in s[:i]:
            continue
        elif s[i] in string.whitespace:
            a += 1
        elif "A" <= s[i] <= "Z":
            b += 1
    return len(s) < 10 and a > 1 and a == b
```

Copy

2. Fix the Style [10 pts]
   We've written a program, areAnagrams(s1, s2), which returns True if two strings are anagrams (that is, if they contain the same letters in possibly-different orders). For example, "smart" and "trams" are anagrams; on the other hand, "read" and "dared" are not anagrams, since they have a different number of letters. The program is functionally correct, but we've written it with terrible style.

   Fix this program so that it meets the 15-112 style guide without rewriting the main logic. Then, in a comment above the fixed program, write out a list of the changes you made to fix the style. You must make valid changes that cover at least five different style rules to get full credit. This problem is not autograded; we will hand-grade it instead.

```
def areAnagrams(s1, s2):
    if len(s1) != len(s2):
        return False
```

```
            print("bad case")
        if len(s1) == len(s2):
            for str in s1:
                one = 1
                count_matches_1= 0
                count_matches_2 = 0
                for i in range(len(s1)):
                    if s1[i] == str:
                        count_matches_1 += one
                for i in range(len(s2)):
                    if s2[i] == str:
                        count_matches_2 += one
                if count_matches_1 !=count_matches_2:
                    return False
            return True
```

Copy

3. Code Writing: applyCaesarCipher(message, shiftNum) [10 pts]
   A Caesar Cipher is a simple cipher that works by shifting each letter in the given message by a certain number. For example, if we shift the message "We Attack At Dawn" by 1 letter, it becomes "Xf Buubdl Bu Ebxo".

   Write the function applyCaesarCipher(message, shiftNum) which shifts the given message by shiftNum letters. You are guaranteed that message is a string, and that shiftNum is an integer between -25 and 25. Capital letters should stay capital and lowercase letters should stay lowercase, and non-letter characters should not be changed. Note that "Z" wraps around to "A". So, for example:

```
assert(applyCaesarCipher("We Attack At Dawn", 1) == "Xf Buubdl B
assert(applyCaesarCipher("zodiac", -2) == "xmbgya")
```

4. COLLABORATIVE Code Writing: gradebookSummary(gradebookFilename) [15 pts]
   For this problem, we'll assume that gradebooks are stored in .txt files. Each row of the gradebook file contains a student's name (one word, all lowercase), followed by one or more comma-separated integer grades. A gradebook always contains at least one student, and each row always contains at least one grade. Gradebooks can also contain blank lines and lines starting with the "#" character, which should be ignored.

   With this in mind, write the function gradebookSummary(gradebookFilename) that takes the filename of a gradebook as an argument and returns a summary of the gradebook as a string. This summary string should show each student followed by a tab followed by their average grade (rounded to the hundreth place). The summary string should have the students listed in their original order (separated by newlines, but without a newline at the end), but should get rid of any comments or blank lines.

   For example, here is a test case:

```
# the following string is the content of the file gradebook1.txt
"""# ignore blank lines and lines starting with #'s
wilma,91,93,94
fred,80,85,90,97,100
betty,88"""
assert(gradebookSummary("gradebook1.txt") == "wilma\t92.67\nfred
```

5. Code Writing: patternedMessage(message, pattern) [15 pts]
   Write the function patternedMessage(message, pattern) that takes two strings, a message and a pattern, and returns a string produced by replacing the non-whitespace characters in the pattern with the non-whitespace characters in the message (where any leading or trailing newlines in

the pattern are first removed), with the message repeating over and over again until the pattern is complete. As a first example:

call	result
patternedMessage("Go Pirates!!!", """ ********** ****** ****** ********** """)	GoPirates!!!GoP irates !!!GoP irates!!!GoPira

Here, the message is "Go Pirates!!!" and the pattern is a block of asterisks with a few missing in the middle. Notice how the whitespace in the pattern is preserved, but the whitespace in the message is removed. Again, note that any leading or trailing newlines in the pattern are removed.

Here is another example:

call	result

patternedMessage("Three Diamonds!","""
  * * *
 *** *** ***
***** ***** *****
 *** *** ***
  * * *
""")
	T h r
	eeD iam ond
	s!Thr eeDia monds
	!Th ree Dia
	m o n

Hint: In our sample solution, we started with an empty result string, then built up the answer character by character. How did we determine the next character? Using both the message and the pattern in some way...

And here is one last example, just for fun:

```
patternedMessage("Go Steelers!",
"""
                 oooo$$$$$$$$$$oooo
              oo$$$$$$$$$$$$$$$$$$$$$$o
           oo$$$$$$$$$$$$$$$$$$$$$$$$$$$o       o$   $$ o$
  o $ oo      o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o      $$ $$ $$o$
oo $ $ '$     o$$$$$$$$$   $$$$$$$$$$$$   $$$$$$$$o      $$$o$$o$
'$$$$$$o$    o$$$$$$$$$     $$$$$$$$$$     $$$$$$$$$o   $$$$$$$
 $$$$$$$   $$$$$$$$$$     $$$$$$$$$$     $$$$$$$$$$$$$$$$$$$$$
 $$$$$$$$$$$$$$$$$$$$$   $$$$$$$$$$$$   $$$$$$$$$$$$$ '$$$
  '$$$'$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$    '$$$
   $$$  o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$    '$$$o
  o$$'  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$      $$$o
  $$$   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$' '$$$$$ooooo$$$$o
 o$$$oooo$$$$$  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$  o$$$$$$$$$$$$$$$$
 $$$$$$$'$$$$  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     $$$$'
 ''''     $$$$   '$$$$$$$$$$$$$$$$$$$$$$$$$$$$'      o$$$
        '$$$o     '$$$$$$$$$$$$$$$$$$'$$'        $$$
         $$$o         '$$'$$$$$$'          o$$$
         $$$$o                          o$$$'
          '$$$$o      o$$$$$$o'$$$$o       o$$$$
           '$$$$$oo     '$$$$o$$$$$o   o$$$$'
             '$$$$$oooo  '$$$o$$$$$$$$$'
              '$$$$$$$oo $$$$$$$$$$
                   '$$$$$$$$$$$
                      $$$$$$$$$$$$
                       $$$$$$$$$'
```

```
                                '$$$'
""")
```

Returns this:

```
                    GoSteelers!GoSteeler
                 s!GoSteelers!GoSteelers!GoS
               teelers!GoSteelers!GoSteelers!GoS        te   el er
  s ! Go       Steelers!GoSteelers!GoSteelers!GoSteel      er s! GoSt
ee l e rs     !GoSteeler   s!GoSteelers!   GoSteelers      !GoSteel
ers!GoSte    elers!GoSt    eelers!GoSt    eelers!GoSt   eelers!G
 oSteele   rs!GoSteele    rs!GoSteele    rs!GoSteelers!GoSteeler
 s!GoSteelers!GoSteelers   !GoSteelers!G   oSteelers!GoSt  eele
  rs!GoSteelers!GoSteelers!GoSteelers!GoSteelers!GoSteel    ers!
   GoS   teelers!GoSteelers!GoSteelers!GoSteelers!GoSteelers    !GoSt
  eele   rs!GoSteelers!GoSteelers!GoSteelers!GoSteelers!GoSt      eele
  rs!   GoSteelers!GoSteelers!GoSteelers!GoSteelers!Go Steelers!GoSteele
 rs!GoSteelers  !GoSteelers!GoSteelers!GoSteelers!GoS   teelers!GoSteelers
 !GoSteelers!G   oSteelers!GoSteelers!GoSteelers!Go     Steel
ers!      GoSt    eelers!GoSteelers!GoSteelers!G      oSte
         elers     !GoSteelers!GoSteelers!        GoS
          teel         ers!GoSteel          ers!
           GoSte                          elers
            !GoSte      elers!GoSteele       rs!Go
              Steelers     !GoSteelers!   GoStee
                lers!GoSte  elers!GoSteeler
                  s!GoSteele rs!GoSteel
                       ers!GoSteele
                          rs!GoSteeler
                           s!GoSteeler
                            s!GoS
```

6. COLLABORATIVE Top-Down Design: mostFrequentLetters(s) [15 pts]
   In this problem, we want you to practice applying the algorithmic thinking strategy of top-down design. We recommend that you solve this problem by using a primary function and at least one well-designed helper function. You get to decide what that helper function should be!

   Write the function mostFrequentLetters(s), that takes a string s, and ignoring case (so "A" and "a" are treated the same), returns a lowercase string containing the letters of s in most frequently used order. (In the event of a tie between two letters, follow alphabetic order, aka normal string comparison.) For example, mostFrequentLetters("We Attack at Dawn") returns "atwcdekn".

   Note that digits, punctuation, and whitespace are not letters! Also note that seeing as we have not yet covered lists, sets, maps, or efficiency, you are not expected to write the most efficient solution. Finally, if s does not contain any alphabetic characters, the result should be the empty string ("").

7. Top-Down Design: longestCommonSubstring(s1, s2) [15 pts]
   In this problem, we want you to practice applying the algorithmic thinking strategy of top-down design. We recommend that you solve this problem by using a primary function and at least one well-designed helper function. You get to decide what that helper function should be!

   Write a function, longestCommonSubstring(s1, s2), that takes two possibly-empty strings and returns the longest string that occurs in both strings (and returns the empty string if either string is empty). For example:

```
assert(longestCommonSubstring("abcdef", "abqrcdest") == "cde")
assert(longestCommonSubstring("abcdef", "ghi") == "") # the empt
```

   If there are two or more longest common substrings, return the lexicographically smaller one

15-112 Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue

(ie, just use "<" to compare the strings). So, for example:

```
assert(longestCommonSubstring("abcABC", "zzabZZAB") == "AB") # 1
```

8. Top-Down Design: justifyText(text, width) [15 pts]
In this problem, we want you to practice applying the algorithmic thinking strategy of top-down design. We recommend that you solve this problem by using a primary function and at least one well-designed helper function. You get to decide what that helper function should be!

   Write the function justifyText(text, width) that takes a string (which may contain various kinds of whitespace) and a width (which you may assume is a positive integer), and returns that same text as a multi-line string that is left- and right- justified to the given line width (except for the last line, which does not need to be justified). For example, consider the following text:

```
text = """\
We hold these truths to be self-evident:  that all men are creat
that they are endowed by their Creator with certain unalienable
that among these are life, liberty, and the pursuit of happiness
```

Copy

   With this text, a call to justifyText(text, 30) would return this text left- and right-justified with 30 characters per line, as such:

```
"""\
We  hold  these  truths  to be
self-evident: that all men are
created  equal;  that they are
endowed  by their Creator with
certain   unalienable  rights;
that  among  these  are  life,
liberty,  and  the  pursuit of
happiness."""
```

Copy

   To solve this problem, we recommend that you follow these three general algorithmic steps.
   1. First, replace all sequences of any kind of whitespace (spaces, tabs, newlines) with a single space. For our example, that creates the following string:

```
"""\
We hold these truths to be self-evident: that all men are c
```

      Copy

   2. Next, break the text into individual lines by repeatedly finding the existing space as far to the right as possible while still allowing the line to remain under the given width restriction. That space will be replaced with a newline ("\n") to create a new string of multiple lines. For our example, that creates the following string:

```
"""\
We hold these truths to be
self-evident: that all men are
created equal; that they are
endowed by their Creator with
certain unalienable rights;
that among these are life,
liberty, and the pursuit of
happiness."""
```

Copy

3. Finally, for each line, add extra spaces from the left to the right to make it fit the given width. For our example, that creates the final result string. Further examples are shown below.

    - Some lines (like the second line above) will already fit the width and won't need to be changed.
    - Other lines will require that you add spaces. Consider the first line in the example above ("We hold these truths to be"). There are 5 total spaces (one between each word), and since the line is 26 characters long, it requires 4 more spaces to pad out to a width of 30. This is done by adding an extra space to each space from left-to-right.
    - Note that you may have to add more than one extra space of padding to each gap (such as the 3 total spaces between "certain" and "unalienable"), there cannot be more than a 1-space difference between any two gaps on a line, and the extra spaces must all be on the left side of the line.

9. Bonus/Optional: getEvalSteps(expr) [3 pts]
   Write the function getEvalSteps(expr), that takes a string containing a simple arithmetic expression and returns a multi-line string containing the step-by-step (one operator at a time) evaluation of that expression. For example, this call:

   getEvalSteps("2+3*4-8**3%3")

   produces this result (which is a single multi-line string):

   ```
   2+3*4-8**3%3 = 2+3*4-512%3
                = 2+12-512%3
                = 2+12-2
                = 14-2
                = 12
   ```

   You are only responsible for legal input as described below:
   - Numbers (which must be integers and non-negative).
   - Operators (which are limited to +, -, *, /, //, %, and **).
   - All operators are binary, so they take two operands. So there are no unary operators, meaning "-5" is not a legal input. For that, you'd need "0-5".
   - In fact, the previous restriction is even stronger: no intermediate value of expr can be negative! So "1-2+3" is not legal, since it would be converted first into "-1+3" which is not legal. So you can safely ignore all such cases.
   - There are no parentheses and no whitespace.

   Here are more instructions and hints:
   - Operators must be evaluated by precedence (** first, then *,/,//,%, then +,-). Equal-precedence operators are evaluated left-to-right.
   - Evaluation proceeds until a single integer with no operators remains after the equals sign.
   - The equal signs must all be stacked directly over each other.
   - In our sample solution, we used very few string methods, just "find" and "isdigit". You may use others, but you should not spin your wheels trying to find that awesome string method that will make this problem remarkably easier, as that method does not exist.
   - For this function, as any other function, you may not use the eval function, so you will have to write your own equivalent simple-eval just for the kinds of simple expressions you will deal with here. Eval is dangerous and should be avoided, as it can lead to serious bugs or security holes.

15-112
Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue