Compsci 101: Test 2

November 13, 2013

Name: ________________________________

NetID/Login: _____________

Community Standard Acknowledgment (signature) ________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>30 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>70 pts.</td>
<td></td>
</tr>
</tbody>
</table>

In writing code you do not need to worry about specifying the proper import statements. Do not 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:  (Express Yourself (5 points))

Part A (4 points)
For each of the following regular expressions, circle all of the strings that it would match.

1. a+b*c+  (note, matches three strings below)
   abcba
   babab
   bacca
   baaab
   aacbb

2. ^[ab]+$  (note, matches two strings below)
   abcba
   babab
   bacca
   baaab
   aacbb

3. b.?b  (note, matches three strings below)
   abcba
   babab
   bacca
   baaab
   aacbb

4. [ab]c[ab]  (note, matches two strings below)
   abcba
   babab
   bacca
   baaab
   aacbb

Part B (1 point)
Write a regular expression that matches any string that may start with the letter d, optionally followed by
the letters a, b, c in any order, and ends with the letter d. For example:

<table>
<thead>
<tr>
<th>MATCHES</th>
<th>DOES NOT MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>abc</td>
</tr>
<tr>
<td>dd</td>
<td>da</td>
</tr>
<tr>
<td>dad</td>
<td>adad</td>
</tr>
<tr>
<td>dabc</td>
<td>ddaad</td>
</tr>
<tr>
<td>baaaad</td>
<td>abced</td>
</tr>
</tbody>
</table>
PROBLEM 2 : (Bad Dates (30 points))

These questions ask you to write code that references a list of strings, representing dates (in the format month then year, MM/YY), whose length is always 7 characters. An example of such a list is given below, but the Python code you write should work with any such values stored in the list dates.

```python
```

In writing code for these questions, you may use any of the variables or functions that have been previously written. You may also assume they contain the correct value regardless of how your code works.

Part A (4 points)

This representation of dates does not allow them to be sorted directly since the month would take precedence over the year. To sort this list, it is necessary to write your own function that compares the relative order of two elements in the list and pass it to `sorted`. This comparison function is given two strings from the list, i.e., two dates, and returns a negative value if the first date given is chronologically before the second date given, 0 if both dates are equal, and a positive value if the first date is chronologically after the second date.

For example, the call `compareDates("11/2013", "01/2016")` returns a negative value; the call `compareDates("11/2013", "01/2012")` returns a positive value; and the call `compareDates("11/2013", "11/2013")` returns 0.

In writing this function, you can return any positive or negative value, since only its sign is checked. Here are two examples of potentially valid return values because both result in a negative value indicating that the third month (March) is chronologically before the eleventh month (November):

```python
int("03") - int("11")  # -8, basic subtraction
cmp("03", "11")  # -1, call python function cmp(stringA, stringB)
```

Sorting the list `dates` given at the beginning of this problem using this comparison function results in the following list:

```python
>>> sortedDates = sorted(dates, cmp=compareDates)
>>> sortedDates
```

Write the function `compareDates` below.

```python
def compareDates(dateA, dateB):
    """
    returns negative int if dateA is chronologically before dateB,
    0 if dateA is equal to dateB,
    positive int if dateA is chronologically after dateB
    given parameters are strings representing dates of the form 'MM/YY'
    """
```
Part B (5 points)
Using the comparison function written in the previous part, write a function that, given a list of strings representing dates and a string representing a date, returns an int, the index where the date would be in the list if it were in proper sorted order (i.e., the index of the first date in the list chronologically equal to or after the new date). If the new date is later than all dates in the list, return the length of the list (i.e., one past the last valid index).

For example, using the sorted version of the list shown in the previous part, the call getSortedIndex(sortedDates, "01/2001") returns 0 since it is before any other date in the list; the call getSortedIndex(sortedDates, "12/2013") returns 11 since it is after all other dates; and the call getSortedIndex(sortedDates, "08/2004") returns 1, since that is the index of the first date it is not later than.

Write the function getSortedIndex below. You may assume the list is already in sorted order and that when you use compareDates to determine the new date's spot in the list, it works correctly regardless of how you wrote it.

```python
def getSortedIndex (sortedDates, dateToInsert):
    """
    returns int, index where the string, dateToInsert, would be in
    the list of strings, sortedDates, if it were in sorted order
    """
```

Part C (5 points)
Write code to store in the variable months the unique strings in the list dates that are the months of each date. For the example list dates this is ['11', '10', '01', '06', '08', '12'] (order of words does not matter).
Part D (7 points)
Write code to store in the variable `monthMax` the string representing the month that occurs most often in the list `dates`. This is "11" in the example list `dates`. Assume the string that occurs most often is unique, i.e., do not worry about breaking ties.

Part E (9 points)
Write code to store in the variable `yearsFreqs` the unique strings in the list `dates` that are the years of each date sorted by number of times each year occurs with the least frequently occurring string first. For strings that occur the same number of times, break ties chronologically (i.e., alphabetically). For the example list `dates` this is ['2005', '2009', '2004', '2008', '2010', '2013'].

The `highCard` function returns the maximum number of points a cheater can win given two lists of integers representing two hands of cards to be played against each other in rounds. A player earns a point by playing the higher card for that round. The cheater knows exactly what cards his opponent has and in what order; the cheater also can order his cards in any way he wants.

This table illustrates what the function is supposed to return:

<table>
<thead>
<tr>
<th>ID</th>
<th>Call</th>
<th>Expected Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>highCard([2, 3, 4, 7], [3, 4, 3, 8])</td>
<td>2</td>
</tr>
<tr>
<td>#2</td>
<td>highCard([3, 3, 2, 2, 5, 3], [3, 4, 2, 9, 8, 2])</td>
<td>3</td>
</tr>
<tr>
<td>#3</td>
<td>highCard([2, 4, 6, 8], [3, 5, 7, 9])</td>
<td>3</td>
</tr>
</tbody>
</table>

Consider the following solutions that are not correct. Each fails at least one of the examples shown above. Complete the table below for each solution, saying whether the given solution passes or fails the referenced test case and, if it fails, what the actual return value is. After the table explain, in general terms, what is wrong with the solution.

```
def highCard (mine, friend):
    points = 0
    for val in reversed(sorted(mine)):
        beatables = [x for x in friend if x < val]
        if len(beatables) > 0:
            points += 1
    return points
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Pass/Fail</th>
<th>Actual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem:

```
def highCard (mine, friend):
    for val in reversed(sorted(mine)):
        beatables = [x for x in friend if x < val]
        if len(beatables) > 0:
            friend.remove(beatables[0])
    return len(mine) - len(friend)
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Pass/Fail</th>
<th>Actual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem:

```
def highCard (mine, friend):
    points = 0
    for m in reversed(sorted(mine)):
        for f in reversed(sorted(friend)):
            if m > f:
                friend.remove(f)
            points += 1
    return points
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Pass/Fail</th>
<th>Actual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem:
PROBLEM 4:  \textit{(Four star rating (20 points))}

These questions ask you about the code from lab 9 regarding data about movies that is given as a handout with the exam. The first two questions ask you to describe how that code works, the last two ask you to write new code using the dictionary \texttt{movies}. In writing code for these questions, you may use any of the variables or functions available in the code.

\textbf{Part A (4 points)}

In the dictionary \texttt{movies}, the values are a list of strings, with some having length 8 and some having length 10. Explain the difference, what does it mean when there are 10 elements of movie data?

Also, for those movies that have 10 elements, what value must the final string represent?

\textbf{Part B (4 points)}

The function \texttt{processCSVdata} takes as its last parameter, \texttt{result}, a dictionary. Explain the purpose of this parameter given its two different values within the code:

\begin{verbatim}
    movies = processCSVdata(rated, 1, 2, {})  
    movies = processCSVdata(rated, 1, 2, movies)
\end{verbatim}
Part C (5 points)
Write a function that returns a list of strings, representing the names of people that both directed and acted in the movies in the given dictionary, in sorted order, alphabetically by person’s first name.

A partial output of the list returned might look like this:

['Charles Chaplin', 'Charles Laughton', 'Clint Eastwood', 'Harold Ramis', ..., 'Woody Allen']

There is already a function that returns the unique directors and here is a function that returns the unique actors for the given movies (it is a simplified version of code you already wrote for the assignment):

```python
def uniqueCastMembers (movies):
    actors = set()
    for (k,v) in movies.items():
        actors.add(v[1:6])
    return actors
```

Write the function `directorAndCast` below.

```python
def directorAndCast (movies):
    ""
    returns a list of strings, the names of directors that also starred in the movies
    """
Part D (7 points)
Write a function that returns a list of tuples of an int and a string, (number of movies, decade), for the movies in the given dictionary, in sorted order by number from highest to lowest.

A partial output of the list returned might look like this:

```
[(157, '2000s'), (75, '1990s'), (72, '2010s'), (42, '1980s'), (31, '1950s'), ..., (4, '1920s')]
```

Note, a nice trick you can use to determine the decade is to ignore the last digit of the year in which the movie was produced:

```
decade = year[:3]
```

and then add the ‘0s’ back in for the tuple to make it easier to read:

```
(count, decade + '0s')
```

Write the function `mostMoviesPerDecade` below.

```python
def mostMoviesPerDecade (movies):
    """
    returns a list of tuples, (number of movies, decade name)
    for the given movies
    """
```