Test 2: Compsci 101

Kristin Stephens-Martinez

November 18, 2019

Name:  

NetID/Login:  

Section Number:  

Honor code acknowledgment (signature):  

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front Page</td>
<td>2 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 1</td>
<td>38 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>84 pts.</td>
<td></td>
</tr>
</tbody>
</table>

**Bubble sheet:** Tear off the bubble sheet from the front of the exam and fill in the Name, ID, and Date field. Your ID is your NetID.

This test has 14 pages be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

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.

**Be sure your name and NetID are legible on this page and that your NetID appears at the top of every page.**

There are two blank pages at the end of the test for extra work space.
PROBLEM 1:  (What will Python display? (38 points))

Part A (26 points)
Choose the output for each print statement.

```python
d = {'e':4, 'n':6, 'o':1}
d['n'] = 4
d['r'] = 2
print(sorted(d.values()))  #1
n = len(set(d.keys()))
m = len(set(d.values()))
print(n == m)  #2
print('r' in d)  #3
print(4 in d)  #4
```

1. What is printed?
   (a) ['e', 'n', 'o', 'r']
   (b) ['e', 'n', 'o']
   (c) [1, 2, 4, 4]
   (d) [1, 4, 6]
   (e) None of the Above

2. What is printed? (a) 3 (b) 4 (c) False (d) True (e) None of the Above

3. What is printed?
   (a) Error (b) False (c) None (d) None of the Above

4. What is printed?
   (a) Error (b) None (c) True (d) None of the Above

```python
set1 = set([8,0,4,8,2])
print(len(set1))  #5
set1.add(5)
set1.add(2)
print(len(set1))  #6
```

5. What is printed? (a) 3 (b) 4 (c) 5 (d) Error (e) None of the Above

6. What is printed? (a) 4 (b) 5 (c) 6 (d) 7 (e) None of the Above

```python
set2 = set([1,2,6])
print(set1 - set2)  #7
print(set1 & set2)  #8
```

7. What is printed?
   (a) {0, 1, 2, 4, 5, 6, 8}
   (b) {0, 8, 4, 5}
   (c) {1, 6}
   (d) {2}
   (e) None of the Above

8. What is printed?
   (a) {0, 1, 2, 4, 5, 6, 8}
   (b) {0, 8, 4, 5}
   (c) {1, 6}
   (d) {2}
   (e) None of the Above
# Reminder: 1/0 causes an Error
print((True and False) or (True or 1/0)) #9  
print((not False) and (False or 1/0)) #10  

9. What is printed? (a) Error (b) False (c) None (d) True (e) None of the Above
10. What is printed? (a) Error (b) False (c) None (d) True (e) None of the Above

lst = [c for c in 'hello' if c in 'good']
print(lst) #11

lst1 = ['red', 'blue', 'green']
lst2 = [w for w in lst1 if 'r' in w]
print(lst2) #12

lst = [x**2 for x in range(1,5)]
print(sorted(lst, key=lambda n : n%4)) #13

11. What is printed?
   (a) ['h', 'e', 'l', 'l', 'o']
   (b) ['o', '0']
   (c) ['o']
   (d) []
   (e) None of the Above

12. What is printed?
   (a) ['blue']
   (b) ['red', 'blue', 'green']
   (c) ['red', 'green']
   (d) ['red']
   (e) None of the Above

13. What is printed?
   (a) [0, 1, 4, 9, 16]
   (b) [0, 4, 16, 1, 9]
   (c) [1, 4, 9, 16]
   (d) [4, 16, 1, 9]
   (e) None of the Above
Part B (12 points)

Given the code below. Choose the output for each print statement.

```python
def mystery(lst):
    i = 0
    while i < len(lst):
        if lst[i] % 2 == 1:
            lst[i] = lst[i] * 2
            i += 1
        else:
            if i == len(lst)-1:
                lst = lst[:-1]
            else:
                lst = lst[:i] + lst[i+1:]
    return lst
```

lstBefore = [1,2,3,4]
lstAfter = mystery(lstBefore)
print(lstBefore) #14
print(lstAfter) #15

14. What is printed?
   (a) Error
   (b) None
   (c) [1, 2, 3, 4]
   (d) [2, 2, 3, 4]
   (e) None of the Above

15. What is printed? (a) Error (b) None (c) [2, 6] (d) [4, 8] (e) None of the Above

lstBefore = [2,5,4,3]
lstAfter = mystery(lstBefore)
print(lstBefore) #16
print(lstAfter) #17

16. What is printed?
   (a) Error
   (b) None
   (c) [2, 5, 4, 3]
   (d) [4, 5, 4, 3]
   (e) None of the Above

17. What is printed? (a) Error (b) None (c) [4, 8] (d) [10, 6] (e) None of the Above

lstBefore = [4,7,5,2]
lstAfter = mystery(lstBefore)
print(lstBefore) #18
print(lstAfter) #19

18. What is printed?
   (a) Error
   (b) None
   (c) [4, 7, 5, 2]
   (d) [8, 7, 5, 2]
   (e) None of the Above

19. What is printed? (a) Error (b) None (c) [8, 4] (d) [14, 10] (e) None of the Above
PROBLEM 2:  (Spot the Bug (12 points))

The following function is an implementation of the Calculator problem in APT Quiz 1 except it only uses addition and subtraction. The function `toTen` takes two lists: `lstOp` and `lstNum`. `lstOp` is a list of math operators as strings: `'+'` (add) or `'-'` (subtract). `lstNum` is a list of integers. The ith operator in `lstOp` should be used with the ith number in `lstNum`. Starting with the number 0, return the number of operations it takes to reach exactly the value 10. If the code runs out of operators or numbers without reaching 10, it should return -1. But this function is buggy! Below are some example calls of what the function should return.

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toTen(['+', [10]])</code></td>
<td>1</td>
</tr>
<tr>
<td><code>toTen(['+', '+'], [5, -5])</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>toTen(['+', '−', '+'], [-2, 8])</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>toTen(['−'], [-5, 5])</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>toTen(['+', '+', '+'], [6, 6, -2])</code></td>
<td>3</td>
</tr>
<tr>
<td><code>toTen(['+', '+', '−', '+', '−'], [2, 7, -3, -2, 4])</code></td>
<td>4</td>
</tr>
<tr>
<td><code>toTen(['−', '+', '−'], [3, 13, 10])</code></td>
<td>2</td>
</tr>
<tr>
<td><code>toTen(['+', '−'], [5])</code></td>
<td>-1</td>
</tr>
</tbody>
</table>

```python
def toTen(lstOp, lstNum):
    i = 0
    val = 0
    while(i<len(lstOp) and val != 10):
        if lstOp[i] == '+':
            val += lstNum[i]
        elif lstOp[i] == '-':
            val -= lstNum[i]
        i += 1
    if val != 10:
        return -1
    return i
```

Part A (4 points)
The following two questions are checkbox questions, a.k.a. multiple options could be the answer.

20. Of the following function calls to the buggy code, which WILL NOT work as expected due to the code’s bug? (There may be more than one)
   (a) `toTen(['+'], [10])` should return 1
   (b) `toTen(['+', '+'], [5, -5])` should return -1
   (c) `toTen(['+', '−', '+'], [-2, 8])` should return -1
   (d) `toTen(['−'], [-5, 5])` should return -1
   (e) None of the above

21. Of the following function calls to the buggy code, which WILL NOT work as expected due to the code’s bug? (There may be more than one)
   (a) `toTen(['+', '+', '+'], [6, 6, -2])` should return 3
   (b) `toTen(['+', '+', '−', '+', '−'], [2, 7, -3, -2, 4])` should return 4
   (c) `toTen(['−', '+', '−'], [3, 13, 10])` should return 2
   (d) `toTen(['+', '−'], [5])` should return -1
   (e) None of the above
Part B (4 points)
In the first two cells below, provide the arguments for a call to `toTen` with either your own arguments or arguments from the examples that returns a `wrong` value. In the “actual return value” cell, write your function call’s return value. If the function call causes an error, write “Error” in the cell. In the “correct return value” cell, write the value it should return.

<table>
<thead>
<tr>
<th>argument: <code>lstOp</code></th>
<th>argument: <code>lstNum</code></th>
<th>actual return value</th>
<th>correct return value</th>
</tr>
</thead>
</table>

Part C (4 points)
Here is the buggy code again. Fix it so that it always returns the correct values. If you reimplement the function, you will earn either all or none of the points for this problem.

```python
i = 0
val = 0

while(i<len(lstOp) and val != 10):
    if lstOp[i] == '+':
        val += lstNum[i]
    elif lstOp[i] == '-':
        val -= lstNum[i]
    i += 1

if val != 10:
    return -1

return i
```
PROBLEM 3 :  *(List Comprehensions (6 points))*

In this problem, you will write code that uses the list variable `people` below. It is a list of lists, where each inner list is of length 3. The values are a person’s first name (string), last name (string), and age (int) respectively. For example, the first element is a person named Vicki Spense that is of age 20.

```python
people = [['Vicki', 'Spence', 20], ['Jane', 'Velasquez', 37],
          ['Chris', 'Estes', 30], ['Frank', 'Gallagher', 15],
          ['Anshul', 'Velasquez', 17], ['Rebecca', 'Estes', 15]]
```

For example, the list comprehension below evaluations to `['Chris', 'Rebecca']`.

```
[p[0] for p in people if p[1] == 'Estes']
```

**Part A (3 points)**

Write code to store in the list variable `drinking` the people that are 21 years old or over. The variable should be a list of strings where each string is the person’s last name then first name separated by a comma. With the list above this would result in the list `['Velasquez, Jane', 'Estes, Chris']`, but the code you write should work for any list named `people` in the format above.

**Part B (3 points)**

Write the value of the list variable `lst` after the list comprehension assigns a value to `lst`.

```
```
PROBLEM 4:  \textit{(Matrices (10 points))}

\begin{verbatim}
matrix = [
    [0, 0, 4, 0],
    [0, 0, 0, 1],
    [0, 3, 0, 0],
]
sparse = {
    (0, 2): 4,
    (1, 3): 1,
    (2, 1): 3,
}
\end{verbatim}

A matrix can be represented in Python as a list of lists (like \texttt{matrix} above), where each inner list represents a row in the matrix. A sparse matrix is a matrix that is mostly 0's (like the one above). A sparse matrix can also be represented as a dictionary. The key is a tuple with the row and column coordinate with 0-based indexing (e.g. the value 4 above is at coordinate (0, 2)). Above \texttt{sparse} is the dictionary version of \texttt{matrix}.

You will be writing code to expand a sparse matrix, represented like \texttt{sparse}, into a full matrix, represented like \texttt{matrix}.

\textbf{Part A (3 points)}

Implement the function \texttt{allZeroMatrix} as described below. Example calls are:

\begin{verbatim}
<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allZeroMatrix(1, 1)</td>
<td>[[0]]</td>
</tr>
<tr>
<td>allZeroMatrix(2, 2)</td>
<td>[[0, 0], [0, 0]]</td>
</tr>
<tr>
<td>allZeroMatrix(2, 3)</td>
<td>[[0, 0, 0], [0, 0, 0]]</td>
</tr>
<tr>
<td>allZeroMatrix(4, 2)</td>
<td>[[0, 0], [0, 0], [0, 0], [0, 0]]</td>
</tr>
</tbody>
</table>
\end{verbatim}

\begin{verbatim}
def allZeroMatrix(n, m):
    """
    n (int) - Height of the matrix, n>0.
    m (int) - Width of the matrix, m>0.

    Return a list of lists of 0's that represent an all zero matrix. The height, number of inner lists, is N. The width, length of the inner lists, is M. The 0's must be independently and individually mutatable.
    """
\end{verbatim}
Part B (7 points)

Implement the function `getMatrix` as described below. Assume your `allZeroMatrix` function works correctly and make sure to use it. You will lose points if you reimplement your `allZeroMatrix` functionality. Example calls are:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getMatrix({(1, 1): 4, (1, 0): 2}, 2, 2)</td>
<td>[[0, 0], [2, 4]]</td>
</tr>
<tr>
<td>getMatrix({(0, 0): 2, (0, 1): 1, (1, 0): 3, (1, 1): 1}, 2, 2)</td>
<td>[[2, 1], [3, 1]]</td>
</tr>
<tr>
<td>getMatrix({(1, 0): 3, (2, 1): 1}, 3, 2)</td>
<td>[[0, 0], [3, 0], [0, 1]]</td>
</tr>
<tr>
<td>getMatrix({(0, 0): 1, (1, 0): 5, (1, 3): 9}, 2, 4)</td>
<td>[[1, 0, 0, 0], [5, 0, 0, 9]]</td>
</tr>
<tr>
<td>getMatrix({(0, 0): 1}, 2, 3)</td>
<td>[[1, 0, 0], [0, 0, 0]]</td>
</tr>
</tbody>
</table>

```python
def getMatrix(sparse, n, m):
    
    sparse (dict) - Keys are coordinate tuples, values are numbers.
    n (int) - Height of the matrix, n>0.
    m (int) - Width of the matrix, m>0.

    Expand the sparse matrix into a full matrix and return the full matrix. Do this by using N and M as the dimensions of the new matrix and the dictionary’s keys as 0-indexed coordinates into the full matrix for the dictionary’s values.
    
    ""
    ```
PROBLEM 5:  (Checkbox Problem Grading (16 points))

Part A (9 points)
Implement the function grade as described below.

Example calls are:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade('a', 'abc', 'a')</td>
<td>3</td>
<td>1pt for correctly including a, 2pts for not including b and c.</td>
</tr>
<tr>
<td>grade('mx', 'mmx', 'mx')</td>
<td>3</td>
<td>2pts for correctly including m and x, 1pt for not including n. Note: Any letters can be used.</td>
</tr>
<tr>
<td>grade('xy', 'xyz', 'z')</td>
<td>0</td>
<td>Exact opposite of correct answer, so no points earned.</td>
</tr>
<tr>
<td>grade('bc', 'abcd', 'ab')</td>
<td>2</td>
<td>Correct for including b and not including d, but incorrect for a and c, so earned 2 of 4 points.</td>
</tr>
</tbody>
</table>

```python
def grade(correct, possible, answer):
    
    correct (str) - The correct options.
    possible (str) - All the possible options.
    answer (str) - The options the student chose.

    Return the points earned for the answer. A student earns 1pt for
each correctly included option in their answer and 1pt for each
correctly not included option in their answer.

    Each option is represented as a single character in the strings.

    Assume: All options that are in CORRECT and ANSWER are in POSSIBLE
```
Part B (7 points)
Implement the function `getMode` as described below. Assume your `grade` function works correctly and make sure to use it. You will lose points if you reimplement your `grade` functionality.

Example calls are:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getMode('a', 'ab', ['a', 'a', 'b', 'ab'])</code></td>
<td>[2]</td>
<td>Scores are [2,2,0,1], mode is 2</td>
</tr>
<tr>
<td><code>getMode('a', 'abc', ['bc', 'bc', 'a', 'ab'])</code></td>
<td>[0]</td>
<td>Scores are [0,0,3,2], mode is 0</td>
</tr>
<tr>
<td><code>getMode('a', 'abc', ['ab', 'ab', 'a', 'abc', 'abc'])</code></td>
<td>[1, 2]</td>
<td>Scores are [2,2,3,1,1], modes are 2 and 1, so return these in sorted order</td>
</tr>
<tr>
<td><code>getMode('a', 'abc', ['bc', 'bc', 'a', 'ab', 'a'])</code></td>
<td>[0, 3]</td>
<td>Scores are [0,0,3,2,3], modes are 0 and 3, so return these in sorted order</td>
</tr>
<tr>
<td><code>getMode('b', 'ab', ['a', 'b', 'ab'])</code></td>
<td>[0, 1, 2]</td>
<td>Scores are [0,2,1], all are equally common, so return all in sorted order</td>
</tr>
</tbody>
</table>

```python
def getMode(correct, possible, lst):
    """
    correct (str) - The correct options.
    possible (str) - All the possible options.
    lst (lst of str) - List of student answers.

    For each student answer in LST, calculate the student’s grade by using the grade function. Then return the mode grade (a.k.a. the most common grade) in a list. If there is more than one mode, return all numbers that are the most common in a list in sorted order.
    """
```
PROBLEM 6 :  (Extra Credit (1 point))

Predict what range your percentage grade will be on the exam. If you are correct, you will earn 1 point of extra credit on this exam, rounding in your favor (e.g. you will earn a point if your score is 94.5% and you choose the 90%-94% range). The front page has a table with the number of points for each problem.

- 95% - 100%
- 90% - 94%
- 85% - 89%
- 80% - 84%
- 75% - 79%
- 70% - 74%
- 65% - 69%
- 60% - 64%
- 55% - 59%
- 50% - 54%
- 45% - 49%
- 40% - 44%
- 35% - 39%
- 30% - 34%
- 25% - 29%
- 20% - 24%
- 15% - 19%
- 10% - 14%
- 5% - 9%
- 0% - 4%