Test 2: Compsci 101

Kristin Stephens-Martinez

March 28, 2019

Name: ___________________________________________ (1/2 pt)

NetID/Login: ___________ (1/2 pt)

Section Number: ___________

Honor code acknowledgment (signature) ________________________________ (1 pt)

<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>23 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>11 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>32 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>81 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 16 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 three blank pages at the end of the test for extra work space.
**PROBLEM 1:**  
*(What will Python display? (23 points))*

**Part A (14 points)**
Write the output for each print statement. **Write the output in the right-column under OUTPUT.**

<table>
<thead>
<tr>
<th>CODE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s = set([0, 9, 9, 10])</code>&lt;br&gt;<code>print(len(s))</code> #1&lt;br&gt;<code>s.add(4)</code>&lt;br&gt;<code>s.add(10)</code>&lt;br&gt;<code>print(len(s))</code> #2</td>
<td>#1&lt;br&gt;#2</td>
</tr>
<tr>
<td><code>t = set([3, 10, 4])</code>&lt;br&gt;`print(s</td>
<td>t)<code> #3: Order does not matter&lt;br&gt;</code>print(s - t)` #4: Order does not matter</td>
</tr>
<tr>
<td><code>l = [c for c in 'blue']</code>&lt;br&gt;<code>print(l)</code> #1&lt;br&gt;<code>a = ['cookie', 'ice cream', 'cake']</code>&lt;br&gt;<code>b = [w for w in a if w[-1] == 'e']</code>&lt;br&gt;<code>print(b)</code> #2&lt;br&gt;<code>n = [x*3 for x in range(8) if x % 2 != 0]</code>&lt;br&gt;<code>print(n)</code> #3</td>
<td>#1&lt;br&gt;#2&lt;br&gt;#3</td>
</tr>
<tr>
<td><code>d = {'a': 3, 'z': 2, 'i': 4}</code>&lt;br&gt;<code>d['q'] = 8</code>&lt;br&gt;<code>d['a'] = 8</code>&lt;br&gt;<code>setK = set(d.keys())</code>&lt;br&gt;<code>setV = set(d.values())</code>&lt;br&gt;<code>print(len(setK))</code> #1&lt;br&gt;<code>print(len(setV))</code> #2&lt;br&gt;<code>print('q' in d)</code> #3&lt;br&gt;<code>print(4 in d)</code> #4</td>
<td>#1&lt;br&gt;#2&lt;br&gt;#3&lt;br&gt;#4</td>
</tr>
<tr>
<td><code>d = {'a': 4, 'b': 2, 'c': 3}</code>&lt;br&gt;<code>l = [d[x] for x in 'cab']</code>&lt;br&gt;<code>print(l)</code> #1</td>
<td>#1</td>
</tr>
<tr>
<td><code># Reminder: 1/0 causes a ZeroDivisionError</code>&lt;br&gt;<code>print((False or False) and (1/0 or True))</code> #1&lt;br&gt;<code>print((False or 1/0) and True)</code> #2</td>
<td>#1&lt;br&gt;#2</td>
</tr>
</tbody>
</table>
Part B (9 points)

Given the code below. Write the output for each **print** statement. **Write the output in the right-column under OUTPUT.**

def stuff(lst, n):
    idx = -1
    for i in range(len(lst)):
        if lst[i] <= n:
            idx = i

    if idx == -1:
        lst = [n] + lst
    elif idx == len(lst)-1:
        lst.append(n)
    else:
        lst = lst[:idx+1] + [n] + lst[idx+1:]

    return lst

<table>
<thead>
<tr>
<th>CODE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>lst = [1, 3, 4]</td>
<td>#1</td>
</tr>
<tr>
<td>ret = stuff(lst, 0)</td>
<td>#2</td>
</tr>
<tr>
<td>print(lst)</td>
<td></td>
</tr>
<tr>
<td>print(ret)</td>
<td></td>
</tr>
<tr>
<td>lst = [3, 6, 8]</td>
<td>#1</td>
</tr>
<tr>
<td>ret = stuff(lst, 5)</td>
<td>#2</td>
</tr>
<tr>
<td>print(lst)</td>
<td></td>
</tr>
<tr>
<td>print(ret)</td>
<td></td>
</tr>
<tr>
<td>lst = [3, 4, 7]</td>
<td>#1</td>
</tr>
<tr>
<td>ret = stuff(lst, 10)</td>
<td>#2</td>
</tr>
<tr>
<td>print(lst)</td>
<td></td>
</tr>
<tr>
<td>print(ret)</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 2:  *(Spot the Bug (11 points))*

The below function checks to see if a list is in order. It returns `True` if each element is greater than any element before it and `False` otherwise. 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>check([1, 2, 3])</td>
<td>True</td>
</tr>
<tr>
<td>check([2, 4, 7])</td>
<td>True</td>
</tr>
<tr>
<td>check([3, 2, 4])</td>
<td>False</td>
</tr>
<tr>
<td>check([2, 4, 3])</td>
<td>False</td>
</tr>
</tbody>
</table>

```
def check(l):
    i = 0
    while i < len(l) and l[i] < l[i+1]:
        i += 1
    if i == len(l):
        return True
    else:
        return False
```

Clarifications:
1. Assume there are no duplicates in the list
2. Assume the len(l) >= 2

**Part A (3 points)**

In the first cell below, write a function call for `check` with either your own arguments or arguments from the examples that returns a correct value. In the second cell, write your function call’s return value.

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
</table>

**Part B (4 points)**

In the first cell below, write a function call for `check` with either your own arguments or arguments from the examples that returns a wrong value. In the second cell, write your function call’s return value. If the function call causes an error, write “Error” in the cell. In the third cell, write the value it should return.

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
<th>correct 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.

```python
def check(l):
    i = 0

    while i < len(l) and l[i] < l[i+1]:
        i += 1

    if i == len(l):
        return True
    else:
        return False
```

PROBLEM 3 :  (Comparing phrases (8 points))

Implement the function `phraseCompare(phrase1, phrase2)` as described below. Assume that the parameters consist of only lower case letters and spaces. Also, spaces should not be counted as a letter.

Example calls are:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>phraseCompare('an', 'on')</td>
<td>(1, 1, 1)</td>
</tr>
<tr>
<td>phraseCompare('hello', 'world')</td>
<td>(2, 2, 3)</td>
</tr>
<tr>
<td>phraseCompare('one fish', 'two fish')</td>
<td>(2, 5, 2)</td>
</tr>
<tr>
<td>phraseCompare('the quick brown fox', 'jumps over the lazy dog')</td>
<td>(9, 6, 11)</td>
</tr>
</tbody>
</table>

def phraseCompare(phrase1, phrase2):
    
    phrase1 (str) - a phrase of lower case letters and spaces
    phrase2 (str) - a phrase of lower case letters and spaces
    
    Return a tuple of length 3. The elements of which are:
    0. The number of letters in PHRASE1 and not in PHRASE2
    1. The number of letters in both PHRASE1 and PHRASE2
    2. The number of letters in PHRASE2 and not in PHRASE1

Clarification:
“The number of UNIQUE letters”
PROBLEM 4:  *(Coins (5 points))*

Implement the function `total(coins, values)` as described below.

Example calls are:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>total(['penny', 'dime'], {'penny': 1, 'nickel': 5, 'dime': 10, 'quarter': 25})</code></td>
<td>11</td>
</tr>
<tr>
<td><code>total(['c', 'c', 'a'], {'a': 1, 'b': 2, 'c': 3})</code></td>
<td>7</td>
</tr>
</tbody>
</table>

```python
def total(coins, values):
    #
    # coins (list of strings) - list of coin names, duplicates allowed
    # values (dict from strings to ints) - dictionary of a coin's name to
    # its value

    Return the total value of the coins listed by name in COINS using
    the dictionary VALUES. The VALUES dictionary's keys are the name
    of the coins and the values are how much that coin is worth.

    Assume all values in COINS is a key in VALUES.
```

```
PROBLEM 5:  (Veterinary Office (32 points))

In this problem, you are writing code to help analyze data for a veterinary office. The code that processes the files has already been written. It returns two things. First, a list of tuples of the format (owner, name, type, year of birth, petID). Second, is a dictionary with the petID as a key to a list of appointments as a value that are sorted chronologically with the earliest appointment first. The appointments are tuples of the format (year, month, day, reason for visit). Month and day in this tuple are represented starting at 0. You may assume for any petID in the list is there is a corresponding key in the dictionary.

An example of the list of pets and dictionary of appointments is:

```
pets = [
    ('Emily', 'Roxy', 'reptile', 2000, 4867),
    ('Ji Yeon', 'Riley', 'cat', 2000, 5969),
    ('Molly', 'Scount', 'amphibian', 1990, 2690),
    ('Jake', 'Ellie', 'bird', 2004, 7489),
    ('Jonathan', 'Chloe', 'dog', 1991, 8845),
    ('Jonathan', 'Ruby', 'dog', 1990, 3539),
    ('Jonathan', 'Bella', 'dog', 2016, 2476)
]

appointments = {
    4867: [
        (2015, 1, 28, 'sick'),
        (2017, 9, 21, 'checkup')
    ],
    5969: [
        (2016, 5, 19, 'checkup'),
        (2019, 4, 12, 'sick')
    ],
    2690: [
        (2018, 2, 22, 'other'),
        (2019, 2, 7, 'sick')
    ],
    7489: [
        (2016, 4, 22, 'checkup')
    ],
    8845: [
        (2016, 3, 1, 'checkup'),
        (2016, 4, 4, 'sick'),
        (2016, 6, 3, 'checkup'),
        (2019, 6, 16, 'sick')
    ],
    3539: [
        (2015, 0, 7, 'sick'),
        (2016, 3, 9, 'checkup'),
        (2018, 3, 6, 'other')
    ],
    2476: [
        (2017, 1, 1, 'other'),
        (2019, 0, 6, 'sick')
    ]
}
```
Part A (3 points)

Write code to store in the list variable `birds` all the pet’s names in the list `pets` that have ‘bird’ as the pet’s type. *You can write one or more lines of code.* In the example list, this would result in the list `[‘Ellie’]`. 
Part B (6 points)

To help compare dates, implement the function `compareDate(yearA, monthA, dayA, yearB, monthB, dayB)` as described below.

Example calls are:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compareDate(2019, 2, 8, 2019, 2, 19)</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>compareDate(2019, 1, 1, 2019, 1, 1)</code></td>
<td>0</td>
</tr>
<tr>
<td><code>compareDate(2019, 1, 1, 2010, 11, 1)</code></td>
<td>1</td>
</tr>
</tbody>
</table>

```python
def compareDate(yearA, monthA, dayA, yearB, monthB, dayB):
    '''
    yearA (int) - Year of the first date
    monthA (int) - Month of the first date
    dayA (int) - Day of the first date
    yearB (int) - Year of the second date
    monthB (int) - Month of the second date
    dayB (int) - Day of the second date
    
    Return -1 if the first date is before the second date. 0 if the dates are exactly the same. 1 if the first date is after the second date.
    '''
```
Part C (14 points)

The veterinarian needs to know which pets have not been seen since a certain date. Implement the function `needCheckup(pets, appointments, year, month, day)` as described below. Remember, the lists inside the dictionary `appointments` are already sorted chronologically with the earliest date first.

For example, the following would happen on the console if `pets` and `appointments` held the data from the example.

```python
>>> needCheckup(pets, appointments, 2018, 2, 13)
[('Emily', 'Roxy'), ('Jake', 'Ellie')]
```

Use the helper function `compareDate` from the prior part. Assume it works as specified, regardless of what you wrote. You will get very little credit if you duplicate the functionality represented by that function.

```python
def needCheckup(pets, appointments, year, month, day):
    '''
    pets (list of tuples) - a list of tuples of the format (owner, name, type, year of birth, petID)
    appointments (dictionary) - a dictionary from a petID to a sorted list of tuples of the format (year, month, day, reason for visit)
    year - year of the date of interest
    month - month of the date of interest
    day - day of the date of interest

    Return a list of tuples of the format (owner, name) for all pets that have not been seen for any reason since the date represented by YEAR, MONTH, DAY. Do not include any pets last seen on that date.
    Assume that all petID’s in PETS are keys in APPOINTMENTS and vice versa.
    '''
```
Part D (9 points)

The veterinarian wants to know which pets are seen more often because they are sick than because of a
regular checkup. There are only three reasons for a visit ['sick', 'checkup', 'other']. Implement the
function usuallySick(pets, appointments) as described below.

For example, the following would happen on the console if pets and appointments held the data from the
example.

```python
def usuallySick(pets, appointments):
    '''
    pets (list of tuples) - a list of tuples of the format (owner, name, type,
    year of birth, petID)
    appointments (dictionary) - a dictionary from a petID to a sorted list of
    tuples of the format (year, month, day, reason for visit)

    Return the entire tuple in PETS, for all pets that have strictly more 'sick'
    appointments than 'checkup' appointments. Do not consider appointments of
    type 'other'.

    Assume that all petID's in PETS are keys in APPOINTMENTS and vice versa.
    '
```

```python
>>> usuallySick(pets, appointments)
[('Molly', 'Scout', 'amphibian', 1990, 2690), ('Jonathan', 'Bella', 'dog', 2016, 2476)]
```
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 get the 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%
extra page
extra page
extrapage