Test 2: CompSci 101

Owen Astrachan and Kristin Stephens-Martinez

April 10, 2018

Name: ____________________________________________

NetID/Login: ____________

Section Number: (01-Astrachan, 02-Stephens-Martinez): ____________

Honor code acknowledgment (signature) ________________________________

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>29 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>13 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>74 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 12 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 net-id are legible on this page and that your net-id appears at the top of every page.**

There is one blank pages page at the end of the test for extra work-space.
**PROBLEM 1:**  (What will Python display? (29 points))

**Part A (20 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>
</table>
| ```py lst = [x for x in range(5) if x >= 3]
print(lst)
lst = [x+3 for x in range(5) if x % 2 == 0]
print(lst)``` | `[3,4], [3,5,7]` |
| ```py lst = ['a', 'b']
lst.append([ 'c', 'd'] )
print(lst)``` | `['a', 'b', ['c', 'd']]` |
| ```py lst = [1, 2]
lst = lst + [3, 4]
print(lst)``` | `[1,2,3,4]` |
| ```py lst = [('sedan', 'car'), ('dumpster', 'truck'), ('hatchback', 'car'), ('cruiser', 'motorcycle')] lst = sorted(lst)
print(lst[0]) lst = sorted(lst, key = operator.itemgetter(1))
print(lst[0])``` | `('cruiser', 'motorcycle'), ('hatchback', 'car')` |
| ```py s1 = set([1, 2, 2, 3, 3, 3])
print(sorted(s1)) s2 = set([2, 3, 4, 5, 5])
print(sorted(s1 | s2)) print(sorted(s1.intersection(s2))) print(sorted(s2 - s1))``` | `[1,2,3], [1,2,3,4,5], [2,3], [4,5]` |
| ```py d = {'bi': 2, 'di': 2, 'tetra': 4, 'hepta': 7}
d['septa'] = 7
print(sorted(d.keys())) setKey = set(d.keys()) setVal = set(d.values())
print(len(setKey) == len(setVal)) print(7 in d)``` | `['bi', 'di', 'tetra', 'hepta', 'septa'], False, False` |
Part B (9 points)
What is the output of each of the print statements below? Write the output after each print statement.

desserts = 'candy donut apple pie jelly beans bonbon brownie cupcake'
d = {}
for dessert in desserts.split():
    first = dessert[0]
    if first not in d:
        d[first] = []
        d[first].append(dessert)

tmp = [(len(v), k, v) for k, v in d.items()]
tmp = sorted(tmp, reverse=True)

print(sorted(d.keys()))

['a', 'b', 'c', 'd', 'j', 'p']

print(d['c'])
['candy', 'cupcake']

print(tmp[0])
(3, 'b', ['beans', 'bonbon', 'brownie'])
**PROBLEM 2:  ** (*Patience and Charity (12 points)*)

The *CharityDonor* APT problem statement is with the exam reference sheet. Two all green solutions are shown below. You’ll be asked questions about these solutions.

**Solution A**

```python
6: def nameDonor(contribs):
7:     names = []
8:     dollars = []
9:     for x in contribs:
10:        data = x.split(":")
11:        name = data[0]
12:        amount = float(data[1])
13:        if name not in names:
14:           names.append(name)
15:           dollars.append(0.0)
16:           dex = names.index(name)
17:           dollars[dex] += amount
18:     mx = max(dollars)
19:     for name in sorted(names):
20:        dex = names.index(name)
21:        if dollars[dex] == mx:
22:           return name
```

**Solution B**

```python
6: import operator
7:
8: def nameDonor(contribs):
9:     d = {}
10:    for x in contribs:
11:       data = x.split(":")
12:       if data[0] not in d:
13:          d[data[0]] = 0
14:          d[data[0]] += float(data[1])
15:    x = sorted(d.items())
16:    y = sorted(x,key=operator.itemgetter(1), reverse=True)
17:    return y[0][0]
```

(problems on next page)
Part A (3 points)
The problem statement says that if more than one donor gives the maximal amount that ties should be broken alphabetically. For Solution A, explain how the code in lines 19-23 ensures that ties for the maximal donor will be broken automatically. Make reference to the code by line number in your explanation.

Looping over sorted(names) on line 20 ensures that keys will be examined in alphabetical order. This means the first donor whose contributions are equal to the maximal amount, checked on line 22, will be returned — and that’s alphabetically first. The max value is calculated on line 19.

Part B (3 points)
The problem statement says that if more than one donor gives the maximal amount that ties should be broken alphabetically. For Solution B, explain how the code in lines 16-18 ensures that ties for the maximal donor will be broken automatically. Make reference to the code by line number in your explanation.

Sorting by keys first on line 16 ensures that ties will be broken using alphabetical ordering since the sort is stable. Sorting by amount and reversing on line 17 ensures that largest donor is listed first, that donor is chosen on line 18 using [0] to get the first/largest donor.
Part C (3 points)
Explain the purpose of each of lines 13, 14, and 15 in Solution A as they relate to solving the problem.

When a name is seen for the first time, the name and a donation of 0.0 are added to the parallel lists: names, dollars (lines 14 and 15). Since the lists are parallel, when donors[dex] updated it will be for the name in names[dex]

Part D (3 points)
Explain the purpose of each of lines 12 and 13 in Solution B as they relate to solving the problem.

The first time name is seen the value in d[name] (line 12), the contribution associated with name in the dictionary, is set to 0.0 (line 13). This value will be updated when name is found again and the contributions updated.
PROBLEM 3:  (Miriam Webster (13 points))

In this problem, you will be asked to write code that uses the dictionary variable `calories` below. The keys for this dictionary are strings, which are the names of a food. Each key’s corresponding value is the number of calories for that food. For example, a bagel has 320 calories and strawberries have 50 calories. You can write one or many lines of code for each question below.

```
calories = {'bagel': 320, 'brie': 85, 'frappuccino': 280, 'graham cracker': 140, 
            'large egg': 70, 'small egg': 60, 'apple': 81, 
            'avocado': 250, 'strawberries': 50}
```

For example, the list comprehension

```
[cc for cc in calories.values() if cc < 100]
```

evaluates to the list [85, 70, 60, 81, 50].

**Part A (4 points)**

Write code to store in list variable `lowcal` the food names (the keys in dictionary in `calories`) that have strictly fewer than 90 calories. In the dictionary above that would be ['brie', 'large egg', 'small egg', 'apple', 'strawberries'], but the code you write should work with any dictionary in the format described.

```
lowcal = [food for food in d.keys() if d[food] < 90]
```

**Part B (3 points)**

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

```
ab = [k for k in calories.keys() if len(k.split()) > 1]
```

`['graham cracker', 'large egg', 'small egg']`
The dictionary `calories` is reproduced below.

```python
calories = {'bagel': 320, 'brie': 85, 'frappuccino': 280, 'graham cracker': 140,
            'large egg': 70, 'small egg': 60, 'apple': 81,
            'avocado': 250, 'strawberries': 50}
```

**Part C (6 points)**

Write code to create a dictionary `dd` in which keys are integer values representing calories in the range [0-99], [100-199], [200-299] and so on where the key 0 represents [0-99], 1 represents [100-199], and in general the integer value `k` represents \([k \times 100, k \times 100 + 99]\). For the dictionary above the code you write should store in `dd` a dictionary equivalent to the one shown below, but the order of the keys doesn’t matter and may be different from what’s shown. Your code should work for any values in `calories`.

```python
dd = {}

for key in calories:
    newval = calories[key] // 100
    if newval not in dd:
        d[newval] = []
    d[newval].append(key)
```
PROBLEM 4:  (Order in the Court (20 points))

Consider the list of world capitals and their latitudes stored as strings as follows.

```
data = ["Paris:48.5", "Berlin:52.3", "Canberra:-35.15", "Reykjavik:64.1", "Nairobi:-1.17"]
```

For example, Paris is at latitude 48.5 North and Canberra is at Latitude 35.15 South. Southern latitudes are stored as negative numbers as part of each string in list `data`.

Part A (4 points)

Write function `list2tuple` that returns a list of tuples in the format (string,float) where the string is the name of a capital and the float is the capital's latitude. The value returned by the call `list2tuple(data)` should be:

```
[("Paris", 48.5), ("Berlin", 52.3), ("Canberra", -35.15),
 ("Reykjavik", 64.1), ("Nairobi", -1.17)]
```

Complete the function below. As shown above, the order of the tuples in the returned list is the same as the order of the corresponding strings in parameter `data`.

```
def list2tuple(data):
    
    data is a list of strings in format "capital:latitude"
    returns list of tuples (string,float) as described
    
    ret = []
    for val in data:
        parts = val.split(":")
        cap = parts[0]
        lat = float(parts[1])
        ret.append( (cap, lat) )

    return ret
```
Part B (4 points)
Write function `north2south` whose parameter is a list of tuples in the format returned by `list2tuple`. The function should return a list in which the same tuples are sorted from north to south, that is in order from the northern most (or largest) latitude to the southern most (or least) latitude.

The list returned by `north2south(list2tuple(data))` should be

```python
["Reykjavik", 64.1], ("Berlin", 52.3), ("Paris", 48.5),
("Nairobi", -1.17), ("Canberra", -35.15)]
```

Complete the function below.

```python
def north2south(data):
    
    """
    data is a list of tuples in format (string,float)
    returns a sorted list of the same tuples ordered from
    northern most/greatest float to southern most/least float
    """

    return sorted(data, key=operator.itemgetter(1), reverse=True)
```
Part C (12 points)
Write the function closest whose parameter is a list of tuples such as that returned by list2tuple. It returns a list of the names of the cities in the parameter tups that are closest in latitude. For the list data shown at the beginning of this problem the call closest(list2tuples(data)) should return either [Berlin,Paris] or [Paris,Berlin] (order of capital cities doesn’t matter).

def closest(tups):
    ""
    tups is a list of tuples in the format (capital,latitude)
    returns a 2-element list of the capitals that are closest
    ""
    mn = 50000
    c1 = ""
    c2 = ""
    for x in range(len(tups)):
        for y in range(x+1,len(tups)):
            t1 = tups[x]
            t2 = tups[y]
            diff = abs(t1[1] - t2[1])
            if diff < mn:
                mn = diff
                c1 = t1[0]
                c2 = t2[0]
    return [c1,c2]
PROBLEM 5 : (Blank Pages)