Test 2: Compsci 101

Kristin Stephens-Martinez

April 7-8, 2020

Name: _____ (1/2 pt)

NetID/Login: _____ (1/2 pt)

	Value	Grade
Front Page	1 pts.	
Problem 1	26 pts.	
Problem 2	14 pts.	
Problem 3	8 pts.	
Problem 4	19 pts.	
TOTAL:	68 pts.	

Put your **name and NetId** on the front page of the pdf you will submit for your exam. That will be either this pdf, if you are going to directly edit it, or the file you will convert to a pdf and submit.

This test has 16 pages be sure your test has them all. This exam is designed to take 75 minutes, but you will have 2.5 hours to do it. Do NOT spend too much time on one question.

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.

PROBLEM 1 : (***Why will Python display this? (26 points)***)

Part A (20 points)

A Python Console using a dictionary is displayed below, illustrating initializing, updating, and examining dictionary keys, values, and items.

```
>>> d = {4: 'e', 1: 't', 9: 'n'}
>>> d[5] = 'f'
>>> d[4] = 'f'
>>> d.keys()
dict_keys([4, 1, 9, 5])
>>> d.values()
dict_values(['f', 't', 'n', 'f'])
>>> d.items()
dict_items([(4, 'f'), (1, 't'), (9, 'n'), (5, 'f')])
```

1. When the statement `print(len(set(d.keys())) == len(d.keys()))` is run after the above is run in the Python Console it prints out `True`. In one or two sentences, explain why.

2. Suppose four new assignment statements of the form `d[key] = value` are executed where `key` and `value` can be anything. Is there any way such assignments result in `print(len(set(d.keys())) == len(d.keys()))` printing `False` rather than `True`? State either yes or no and briefly justify your answer.

3. In the code earlier, the keys in the dictionary all have the same type, but this is not required in Python. It is possible to write `d['f'] = 5`, for example after the statements shown above. The code below shows that the set of keys and values in an empty dictionary are the same.

```
>>> d2 = {}
>>> set(d2.keys()) == set(d2.values())
True
```

Is it possible for the expression `set(d2.keys()) == set(d2.values())` to have the value `True` after two assignments of the form `d2[x] = y` are executed when `d2` is initially empty? Write either yes or no and briefly justify your answer.

4. The line `(False and 1/0)` evaluates to `False` despite the code `1/0`, which would cause a DivideByZero error. In one or two sentences, explain why.

5. The below code executed without errors in the Python Console. This is despite the use of tuples, which are immutable. In one or two sentences, explain why.

```
>>> t = ('a', 2, ['b', 3])
>>> t[-1][0] = 'c'
>>> t
('a', 2, ['c', 3])
```

Part B (6 points)

The code below is executed by a Python interpreter and results in printing three lines as shown to the right.

Code that executes

```
num = 1

def mystery1():
    global num
    num += 1
    return num

def mystery2():
    num = 99
    return num

def mystery3():
    global num
    return num

if __name__ == '__main__':
    print(mystery1())
    print(mystery2())
    print(mystery3())
```

What is printed

```
2
99
2
```

1. Explain why the value returned by `mystery1` is 2.

2. As shown above the value returned by the function `mystery1` is 2 and the value returned by `mystery3` is also 2. Explain why this is the case even though `mystery2` returns 99.

PROBLEM 2 : (*Spot the Bug (14 points)*)

The following function, counts the frequency of the numbers in a list of positive (> 0) numbers. It counts the frequency of all numbers between 1 and 5 inclusively until it sees a number greater than 5 or reaches the end of a list. It then returns the most frequent number between 1 and 5, breaking ties by returning the smallest number. But this function is buggy! Below are some example calls of what the function actually returns versus what it should return.

Function call	Actual value	Correct value
`frequency([1])`	`1`	`1`
`frequency([4, 99])`	`4`	`4`
`frequency([3, 52, 5])`	`3`	`3`
`frequency([2, 3, 5, 3])`	`list index out of range`	`3`
`frequency([1, 2, 3, 4, 5])`	`list index out of range`	`1`
`frequency([1, 3, 1, 3])`	`1`	`1`
`frequency([3, 4, 5, 4, 6, 5])`	`list index out of range`	`4`
`frequency([3, 4, 4, 12, 3])`	`4`	`4`

```
1  def frequency(lstNum):
2      counts = [0 for x in range(5)]
3
4      i = 0
5      while i < len(lstNum) and lstNum[i] <= 5:
6          num = lstNum[i]
7          counts[num] += 1
8          i += 1
9
10    return counts.index(max(counts))
```

Part A (4 points)

In the first cell below, provide the argument for a call to `frequency` with arguments not in the above examples that returns a *correct* value. In the "correct return value" cell, write the value it should return.

argument: `lstNum`	correct return value

Part B (6 points)

In the first cell below, provide the argument for a call to `frequency` with arguments not in the above 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.

argument: `lstNum`	actual return value	correct return value

Part C (4 points)

In two to four sentences, explain what the bug is, how to fix it, and why that fix works.

PROBLEM 3 : (*List Comprehensions (8 points)*)

In this problem, you will write code that uses the dictionary variable `wallArt` below. The dictionary's keys are strings that are the name of a piece of wall art. The dictionary's values are tuples with two elements. The first element is a string representing the wall art's type. The types of each work is one of the strings `'fabric'`, `'paper'`, `'poster'`, or `'painting'`. The second element is an int representing the wall art's price. For example, in the dictionary below the 'Mona Lisa' is a `'painting'` with a cost of $850,000,000.

```
wallArt = {
    'Mona Lisa': ('painting', 850000000),
    'Mona Lisa poster': ('poster', 15),
    'origami cranes': ('paper', 5),
    'tie dye': ('fabric', 10),
    'Queen poster': ('poster', 17),
    'DIY painting': ('painting', 20),
    'paper flowers': ('paper', 12),
    'macrame hanging': ('fabric', 60),
    'Duke poster': ('poster', 25),
}
```

For each of the problems below you can write a list comprehension or a loop to solve the problem. For example, consider the problem of creating a list named `fabricArt` of all wall art names (dictionary keys) that have type `fabric`. For the dictionary above this should be `['tie dye', 'macrame hanging']`, but your code should work for any dictionary named `wallArt` in the format above.

Using a list comprehension you could write:

```
fabricArt = [w for w in wallArt if wallArt[w][0] == 'fabric']
```

Using a loop you could write:

```
fabricArt = []
for w in wallArt:
    tup = wallArt[w]
    if tup[0] == 'fabric':
        fabricArt.append(w)
```

(continued)

Part A (4 points)

```
wallArt = {
    'Mona Lisa': ('painting', 850000000),
    'Mona Lisa poster': ('poster', 15),
    'origami cranes': ('paper', 5),
    'tie dye': ('fabric', 10),
    'Queen poster': ('poster', 17),
    'DIY painting': ('painting', 20),
    'paper flowers': ('paper', 12),
    'macrame hanging': ('fabric', 60),
    'Duke poster': ('poster', 25),
}
```

Write code to store in the list variable `paperOnly` the wall art names (the dictionary's keys) that are of type `poster` or `paper`.

With the dictionary above, the value of `paperOnly` would result in the list `['Mona Lisa poster', 'origami cranes', 'Queen poster', 'paper flowers', 'Duke poster']`, but the code you write should work for any dictionary named `wallArt` in the format above.

Part B (4 points)

```
wallArt = {
    'Mona Lisa': ('painting', 850000000),
    'Mona Lisa poster': ('poster', 15),
    'origami cranes': ('paper', 5),
    'tie dye': ('fabric', 10),
    'Queen poster': ('poster', 17),
    'DIY painting': ('painting', 20),
    'paper flowers': ('paper', 12),
    'macrame hanging': ('fabric', 60),
    'Duke poster': ('poster', 25),
}
```

Write code to store in the list variable `affordable` the wall art types (the first value in the dictionary's value's tuple) that cost less than or equal to $20. The variable should be a list of strings. With the list above this would result in the list `['poster', 'paper', 'fabric', 'poster', 'painting', 'paper']`, but the code you write should work for any dictionary named `wallArt` in the format above.

PROBLEM 4 : (*Gradebook (19 points)*)

In a course each student completes the same number of assignments and the grades on these assignments are stored in a list of `ints` for each student. Another list stores all of these lists. For example, in a course with four students completing five assignments all the grades would be stored as follows:

```
grades = [
            [2, 5, 5, 2, 3],
            [5, 4, 5, 1, 5],
            [1, 4, 3, 5, 2],
            [2, 4, 5, 5, 4]
         ]
```

Note that each inner list has the same number of values (five, one for each assignment) since each student has a grade for every assignment (some might earn a grade of zero).

In the three functions for this problem a `list of list of ints` in this format is in *proper format* when every inner list (sublist) has the same length – e.g., the list does represent grades for students in a course where each student has a grade for each assignment.

Part A (7 points)

Implement the function `getSlice` as described below.

Example calls given the `grades` variable above are:

function call	return value
`getSlice(grades, 0)`	`[2, 5, 1, 2]`
`getSlice(grades, 2)`	`[5, 5, 3, 5]`

```
def getSlice(grades, index):
    '''
    grades (list of lists of ints) - grade list in proper format
    index (int) - index to a particular assignment across all students

    Return all the grades for a specific assignment that is at location INDEX
    within each student's grade list (GRADES' inner lists). Each student should
    contribute one number to the returned list.
    '''
```

Part B (7 points)

To figure out a student's overall grade you need to know the total possible points across all of the assignments. Implement the function `getTotalPossiblePoints` as described below to get this value. Assume each assignment has a student that earned the highest possible score. Assume your `getSlice` function works correctly and *make sure to use it*. You will lose points if you reimplement the `getSlice` functionality.

Example calls are:

Function call	Return value	Note
`getTotalPossiblePoints([[5, 5]])`	`10`	

`getTotalPossiblePoints([`
`  [1, 2, 3],`
`  [2, 1, 3]`
`])`
	`7`	The max score for each column is 2, 2, 3 respectively, which sums to 7.

`getTotalPossiblePoints([`
`  [2, 5, 5, 2, 3],`
`  [5, 4, 5, 1, 5],`
`  [1, 4, 3, 5, 2],`
`  [2, 4, 5, 5, 4]`
`])`
	`25`	The max score for each column is 5. With 5 assignments, the total possible points is 25.

```
def getTotalPossiblePoints(grades):
    '''
    grades (list of lists of ints) - grade list in proper format

    Return the total possible points across all the assignments. This could then be
    used as the denominator to calculate the overall grade. Do this by finding the
    max of each column and returning the sum of those values.

    Assume: All values in GRADES are >= 0
    Assume: At least 1 student in each column earned the max possible points for
        that assignment.
    '''
```

Part C (5 points)

A student's grade is determined by the sum of their assignment scores divided by the total possible points, i.e., divided by the value returned by `getTotalPossiblePoints`.

Implement the function `getStudentGrades` as described below. Assume your `getTotalPossiblePoints` function works correctly and *make sure to use it.* You will lose points if you reimplement the `getTotalPossiblePoints` functionality. Note: The examples are simple to make the math easier to understand. Your code should work for any set of grades that still fit the assumptions.

Example calls are:

Function call	**Return value**	**Note**

`getStudentGrades([`
`  [2, 2, 3],`
`  [0, 5, 3]`
`])`
	`[0.7, 0.8]`	The total possible points is 10 and the student's scores sum to 7 and 8 respectively

`getStudentGrades([`
`  [0, 2],`
`  [2, 1],`
`  [2, 2],`
`  [1, 1],`
`  [1, 0]`
`])`
	`[0.5, 0.75,`
	` 1.0, 0.5,`
	` 0.25]`
		The total possible points is 4 and the student's scores sum to 2, 3, 4, 2, and 1 respectively.

```
def getStudentGrades(grades):
    '''
    grades (list of lists of ints) - grade list in proper format

    Return a list of floats where the ith element is the overall grade of the ith
    student in GRADES. The overall grade is calculated as the total of all of their
    grades divided by the total possible points for all the assignments.

    Assume: All values in GRADES are >= 0
    Assume: At least 1 student in each column earned the max possible points for
        that assignment.
    '''
```

extrapage