15-112
Spring 2019 Midterm 2
April 4, 2019

Name:

Andrew ID:

Recitation Section:

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may use the last page and the margins for scrap work.
- You may complete the problems in any order you'd like; you may wish to start with the free response problems, which are worth most of the credit.
- All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.

Don't write anything in the table below.

Question	Points	Score
1	10	
2	10	
3	10	
4	25	
5	25	
6	20	
7	0	
Total:	100	

1. **Short Answer**

Answer each of the following *very briefly*. For multiple choice questions, select the **best** answer, and **fill in the bubble completely**.

(a) (2 points) The following four questions have to do with hash functions and sets. Choose only **one** answer for each.

(i) What is a hash function?

◯ A function that turns a mutable data type into an immutable data type

◯ A function that turns an immutable value into an integer

◯ A function that places an object into a set or dictionary

◯ A function that finds a value in a list in constant time

(ii) How is a hashed item put into a set?

◯ The hash value tells the set where to store the item in its hidden list, and the set places the item at that location

◯ The hash value is placed into the set instead of the item

◯ The hash value is stored as a key and the item is stored as a value, just like in a dictionary

◯ A set is a list of tuples, where each tuple contains the hash value and the item

(iii) How is it possible to see if an item is in a set in O(1) time?

◯ Since a set has a constant size, we can scan the entire set in O(1) time

◯ We can hash the item again, and it's faster to look up the hash value than the original item, allowing us to search the set in O(1) time

◯ Hashing the item will tell us which index to look into, and we can check that index in O(1) time

◯ We can't see if an item is in a set in O(1) time

(iv) Which of the following is a true statement about hashing and sets?

◯ Sets can be hashed and stored inside of sets

◯ Sets cannot be stored in lists

◯ For a given hashable value, the hash function will return the same output every time.

◯ No two unique values hash to the same result

(b) (4 points) For each of the following two questions, what is the simplified Big-O runtime of the function in terms of N? Write your answer in the box under the code. Write your intermediate work on the lines next to the functions for partial credit.

Built-in Big-O Runtimes	
String/List of length N	
`len(item)`	$O(1)$
`item[i]`	$O(1)$
`print(item)`	$O(N)$
`item.find(x)`	$O(N)$
`item.count(x)`	$O(N)$
`x in item`	$O(N)$
`copy.copy(item)`	$O(N)$

```
import string
def big01(s): # N is the length of the string s
    x = 0                                          # ________
    for i in range(0, len(s), 2):                  # ________
        if s.find(str(i)) != -1:                   # ________
            x += s.count(str(i))                   # ________
        elif s[i] in string.ascii_letters:         # ________
            x = 0                                  # ________
    return x                                       # ________
```

```
import copy
def big02(lst): # N is the length of the list lst
    n, i, result = len(lst), 1, ""                 # ________
    grid = [ copy.copy(lst) for i in range(n) ]    # ________
    row, col = 0, 1                                # ________
    while row < n:                                 # ________
        print(grid)                                # ________
        result += grid[row][col]                   # ________
        i = i * 2                                  # ________
        row, col = i // n, i % n                   # ________
    return result                                  # ________
```

(c) (1 point) The following two questions have to do with sorting algorithms. Choose only **one** answer for each.

(i) Which of the following statements is true regarding bubble sort and selection sort?

◯ They both have to make the same (or approximately the same) number of comparisons and swaps

◯ Bubble sort is O(n**2) while selection sort is O(n log(n))

◯ Bubble sort and selection sort have to make the same (or approximately the same) number of swaps, but bubble sort often performs fewer comparisons

◯ They both have to make the same (or approximately the same) number of comparisons, but selection sort often performs fewer swaps

(ii) Which answer **best** describes merge sort?

◯ It merges every element into a set, where we can quickly find the smallest value in O(1) time

◯ It only runs in O(n log(n)) time if the elements of the list are already mostly ordered

◯ It breaks the list into recursively smaller lists, orders those, and begins merging them until the elements form a single sorted list.

◯ Merge sort returns an ordered list with all duplicate items removed

(d) (2 points) Write the **single-expression function** `oddFactors(x)` in the box below, which returns a list of every odd factor of x, including 1 and itself. You must use at least **two** of the following concepts: `map()`, `filter()`, `reduce()`, `lambda`, and list comprehensions (even if you could do it with only one).

(e) (1 point) Recall the `listFiles(path)` function from Week 10, which recursively returns a list of all files in the folder specified by the string path (as well as its subfolders). We have provided the function below, with two incomplete statements. **Write the code needed to complete these statements in the boxes below**.

```
def listFiles(path):
    if ___???___:  #<------------------Answer in box 'i'
        return [ path ]
    else:
        files = [ ]
        for ______???______: #<--------Answer in box 'ii'
            files += listFiles(path + "/" + filename)
        return files
```

(i) Complete the statement `if ___???___:` in the box below

(ii) Complete the statement `for ______???______:` in the box below

2. **Code Tracing**

Indicate what each piece of code will print. Place your answer (and nothing else) in the box below each piece of code.

(a) (5 points) CT 1

```
def ct1(lst,x):
    L=0
    R=len(lst)-1
    while L<=R:
        i=(L+R)//2
        if lst[i]==x:
            return i
        elif lst[i]<x:
            L=i+1
        else:
            R=i-1
        print(lst[L:R+1])
    return -1

lst=[1,2,3,5,7,8,10,12,12,14,17]
print(ct1(lst,11))
```

(b) (5 points) CT 2

```
def ct2(s, depth=0):
    if len(s) == 0:
        return ""
    else:
        result = ct2(s[1:], depth+1)
        if s[0] in "aeiou":
            result = s[0] + result
        elif s[0] not in result:
            result = result + s[0]
        print(depth, "out", result)
        return result

print(ct2("start"))
```

3. **Reasoning Over Code**

For each function, find parameter values that will make the function return True. Place your answer (and nothing else) in the box below each block of code.

Note: in the second problem, you can get partial credit if you can't figure out the correct answer. See the note above the problem to determine how. If your answer in the left box is correct, you'll get full points; if it's wrong, we'll look at the partial credit box.

(a) (5 points) ROC 1

```
def roc1(lst):
    assert(len(lst) == len(set(lst)) == 3)

    d = { "other" : 0 }
    for item in lst:
        (a, b) = item
        if type(b) == int:
            d[a] = b
        else:
            d["other"] += 1

    return set(d.values()) == { 1, 4 }
```

Answer:

(b) (5 points) ROC 2

Partial Credit: Write in the right-side box what is printed when the function is run on the correct input.

```
def foo(x):
    if x == 1:
        return []
    y = bar(x)
    print(x, y)
    r1 = foo(x//y)
    return r1 + [y]

def bar(n):
    for f in range(2, n+1):
        if n % f == 0:
            return f
    return None

def roc2(x):
    return foo(x) == [5,3,3,2]
```

Answer:

Partial Credit:

4. (25 points) **Free Response: Play and Musical Classes**

Write the classes Play and Musical so that the following test code runs without errors. Do not hardcode against the values used in the testcases. You must use proper object-oriented design, including good inheritance, or you will lose points!

```
# A play has a name and a number of scenes
p1 = Play("Our Town", 34)
assert(str(p1) == "Play<Our Town,34>")
p2 = Play("The Crucible", 5)
assert(str(p2) == "Play<The Crucible,5>")
# A play begins on Scene 1, and can change scenes
assert(p1.getStatus() == "On Scene 1")
p1.sceneChange()
assert(p1.getStatus() == "On Scene 2")
# When it reaches the end, it should say so
for i in range(32):    p1.sceneChange()
assert(p1.getStatus() == "On Scene 34")
p1.sceneChange()
assert(p1.getStatus() == "Show is done")

assert(p1 == Play("Our Town", 34) and p1 != Play("Almost, Maine", 34))
assert(p1 != "Our Town")
s = { Play("Our Town", 34) }
assert((p1 in s) and (p2 not in s))

# A Musical is a Play that has a song list:
# a sorted list of scenes where songs occur
m1 = Musical("Beauty and the Beast", 14, [2,5,6,7,8,10,11,12,13,14])
assert(str(m1) == "Play<Beauty and the Beast,14>")
# A musical can do everything a play can do - with music!
assert(m1.getStatus() == "On Scene 1")
m1.sceneChange()
assert(m1.getStatus() == "On Scene 2, with music!")

# A musical can also skip forward to the next scene with a song
m2 = Musical("Test Case: The Musical", 12, [4,9])
m2.skipToSong()
assert(m2.getStatus() == "On Scene 4, with music!")
m2.sceneChange()
assert(m2.getStatus() == "On Scene 5")
m2.skipToSong()
assert(m2.getStatus() == "On Scene 9, with music!")
# If you're on the last musical scene, don't skip forward!
m2.skipToSong()
assert(m2.getStatus() == "On Scene 9, with music!")
```

Additional Space for Answer to Question 4

Additional Space for Answer to Question 4

5. (25 points) **Free Response: Circle Animation**

Write the functions `init(d)`, `keyPressed(e,d)`, `timerFired(d)`, and `redrawAll(c,d)` for the time-based animation that meets the requirements below. Assume that the run function is defined as in the notes (with a timerDelay of 100), and that we do not need `mousePressed(e,d)`.

The animation has the following properties:

- Every 2 seconds, a circle with radius 10 is generated at a random x and y location.
- The animation draws a 40x40 square in the center of the canvas. It should be drawn such that it does not obscure the circles (they are always visible).
- A number is also drawn in the center of the box. The number is initially 0.
- Every 100ms, each circle should move 5 pixels to the right. If the center of the circle moves off the right edge of the screen, the circle should wrap around to the left side again.
- Every 5 seconds, any circle entirely within the square should disappear.
- For every circle that disappears, the number in the box should increase by 1.
- When the "p" key is pressed, the animation pauses (i.e. no circles should appear, disappear, or move and the number should not change.) Pressing "p" again unpauses the animation.

Note: Anything not specified here is up to you. However, we highly recommend that you don't use OOP. Also, you must follow the MVC framework.

Hint: You should use d, e, and c in place of data, event, and canvas to save time!

Additional Space for Answer to Question 5

Additional Space for Answer to Question 5

6. (20 points) **Free Response:** `combineInts(lst)`

Write the function `combineInts(lst)` which takes a list of integers. The function returns the set of integers which can be created by either adding or subtracting each element of `lst`. Each element must either be added or subtracted, and each element is only used once. For example, the integers in `lst=[1, 1, 2]` can be combined in the following ways:

```
#Possible combinations of 1, 1, and 2
 1 + 1 + 2 ==  4
 1 + 1 - 2 ==  0
 1 - 1 + 2 ==  2
 1 - 1 - 2 == -2
-1 + 1 + 2 ==  2
-1 + 1 - 2 == -2
-1 - 1 + 2 ==  0
-1 - 1 - 2 == -4

#Since we're storing these combinations in a set, this test should pass
assert(combineInts([1, 1, 2]) == {-4, -2, 0, 2, 4})
```

Note: For a list of 3 elements, there are 8 possible ways to add and subtract each integer, but some of those combinations yield the same total, so `combineInts([1, 1, 2])` returns a set of 5 integers. Here are some additional test cases:

```
assert(combineInts([5]) == {-5, 5})
assert(combineInts([1, 5]) == {-6, -4, 4, 6})
assert(combineInts([1, 5, 10]) == {4, 6, 14, -16, 16, -14, -6, -4})
assert(combineInts([]) == {0})
```

Your function must use recursion! You may use iteration inside your recursive function, however.

Additional Space for Answer to Question 6

Additional Space for Answer to Question 6

7. (0 points) **Bonus CT:** Only try this if you're done with the other questions and are bored! Write in the box below what the following code prints. You can get 2 extra points for a correct answer.

```
import functools
def bonus(n):
    g = lambda f: lambda x: f(x) // 2 + 1
    def h(y, fn = lambda z: z - 1):
        return [] if fn(y) < 5 else [fn(y)] + h(y//2, g(fn))
    L = h(n)
    return functools.reduce(lambda x, y: x * 10 + y % 10, L, 0)
print(bonus(112))
```

Use this page for scrap work. Nothing on this page will be graded!

Use this page for scrap work. Nothing on this page will be graded!