15-112
Spring 2018 Midterm Exam 2
April 12, 2018

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 backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- You may complete the problems in any order you'd like; you may wish to start with the last three problems, which are worth most of the credit.
- All code samples run without crashing. Assume any imports are already included as required.

Don't write anything in the table below.

Question	Points	Score
1	10	
2	10	
3	8	
4	12	
5	15	
6	5	
7	20	
8	20	
Total:	100	

1. **Code Tracing** Indicate what each will print. Place your answer (and nothing else) in the box below each block of code.

   (a) (5 points) CT1

```
class A(object):
    def __init__(self,x):
        self.x = "A:" + x
    def __repr__(self):
        return "A(" + str(self.x) + ")"
    def append(self, n):
        self.x += str(n)
        return self.x

class B(A):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = int(y)

    def sub(self):
        self.x = self.x[:-self.y]
        return self

def ct1():
    try:
        a, b = A("q"), B("r", "2")
        print(a)
        print(b)
        print(b.append(6))
        print(b.sub())
        print("All good")
    except:
        print("Uh oh!")
ct1()
```

(b) (5 points) CT2

```
def fRetG(x):
    print("fRetG")
    return x // 2

def myFunc(x):
    print(x)
    if x < 0:
        return 0
    elif x == 1:
        return 1
    else:
        return x + myFunc(x - 5)

def ct2(x):
    print("ret =", myFunc(fRetG(x)))

ct2(32)
```

2. **Reasoning Over Code**

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

   (a) (5 points) ROC1

```
def roc1(d):
    if not isinstance(d, dict):
        return False
    x, y, z = 0, 0, 0
    for k1 in d:
        if isinstance(d[k1], dict):
            x += 1
            for k2 in d[k1]:
                y += 1
        else:
            if k1 != d[k1] and d[k1] in d:
                z += 1
    return (x, y, z) == (1, 2, 1)
```

   (b) (5 points) ROC2

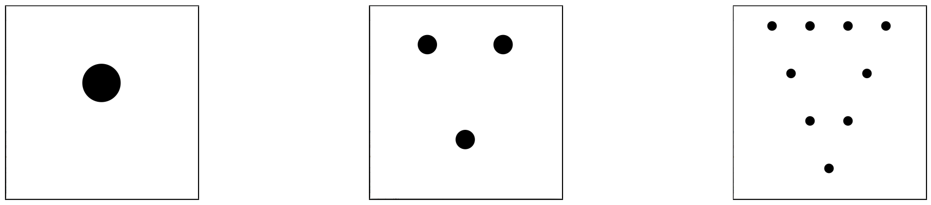
```
def helper(lst):
    if len(lst) == 2:
        return lst
    else:
        left = helper(lst[0:len(lst):2])
        right = helper(lst[1:len(lst):2])
        return left + right
def roc2(lst):
    if len(lst) != len(set(lst)):
        return False
    result = helper(lst)
    return len(result) == 8 and sorted(result) == result
```

3. **Short Answer**

   Answer each of the following *very briefly.*

   (a) (2 points) What are the key steps of adding something to a set that makes that process take O(1) time? Please include key terms.

   (b) (2 points) Below is shown a Dot Fractal at depth 0, 1, and 2. Assume that the function drawDots is called to draw this fractal. In the code, how many calls to drawDots and/or to canvas.create_oval are needed in the base case? How many in the recursive case?



Base Case drawDots: **Recursive Case drawDots:**
Base Case create_oval: **Recursive Case create_oval:**

   (c) (2 points) What is the difference between a class and an instance? Write a small amount of code that creates a class and code that creates an instance.

   (d) (2 points) Say you have an unsorted list lst that contains N elements. Is it possible to determine if lst contains an arbitrary item in less than O(N) runtime? If yes, how? If no, why not?

4. (12 points) **Big-O:** What is the Big-O runtime of each of the following in terms of N?

Built-in Big-O Runtimes					
Sets		Strings		Lists	
`len(S)`	$O(1)$	`len(s)`	$O(1)$	`len(lst)`	$O(1)$
`item in S`	$O(1)$	`c in s`	$O(N)$	`item in lst`	$O(N)$
`S.add(item)`	$O(1)$	`ord(s)`	$O(1)$	`lst[i]`	$O(1)$
`S.remove(item)`	$O(1)$			`lst.sort()`	$O(NlogN)$

```
# S is a set of N numbers
# T is a string of length N
def bigOh1(S, T):                # Big-O
    count = 0                    # _____
    for i in range(len(T)):      # _____
        if ord(T[i]) in S:       # _____
            count += 1           # _____
    return count                 # _____
```

```
# L and M are lists of N integers
def bigOh2(L, M):                # Big-O
    L.sort()                     # _____
    M.sort()                     # _____
    S = set()                    # _____
    for i in range(len(L)):      # _____
        S.add(L[i])              # _____
    for x in M:                  # _____
        if x in L:               # _____
            S.remove(x)          # _____
    return S                     # _____
```

```
# N is an integer
def bigOh3(N):                   # Big-O
    count = 0                    # _____
    while (N > 0):               # _____
        count += 14              # _____
        N = N // 3               # _____
    return count                 # _____
```

5. (15 points) **Free Response: Moving Animation**

   Assuming the run() function is already written for you, write init, keyPressed, mousePressed, redrawAll, and timerFired so that an animation is created with the following properties:

   1. A green circle is drawn centered in the screen with an initial radius of 100 pixels.
   2. The green circle's radius increases by 10 pixels every 0.25 seconds until the circle hits the edge of the screen; then the radius decreases by 10 pixels every 0.25 seconds until it gets down to 0; then it starts increasing again, and etc.
   3. Every second a single purple square is added to the screen with a randomly chosen position and size (where the size can vary from 1 pixel to 50 pixels).
   4. When the user presses 'p', the animation is paused (the circle stops moving and squares stop appearing); when the user presses 'p' again, the animation resumes.

   To save time, you may wish to abbreviate data as d, event as e, and canvas as c.

Additional Space 1 for Answer to Question 5

Additional Space 2 for Answer to Question 5

6. (5 points) **Free Response: mostPopularCities**

   Write the function mostPopularCities(cityMap) which takes a dictionary mapping people (strings) to sets of the cities they've visited (also strings) and returns a set of the cities that the most people have visited. For example, if we provide the following city map:

```
cityMap = { "Abhi": { "Denver", "London" },
            "Arman": { "Pittsburgh", "London" } }
```

   The function would return the set `{ "London" }`, since ”London” was visited the most. If multiple cities tie for most visited, they should all be included in the returned set; if no cities were visited, an empty set should be returned.

7. (20 points) **Free Response: Vehicle and Buggy**

Write the classes Vehicle and Buggy so that the following test code runs without errors. Do not hardcode against the values used in the testcases, though you can assume the testcases cover the needed functionality. You must use proper object-oriented design, including good inheritance, or you will lose points!

```
# a vehicle has a name and a capacity
v1 = Vehicle("Paddleboat", 2)
assert(str(v1) == "Paddleboat holds 2: []")
# passengers can be added
assert(v1.addPassenger("Ken") == True)
assert(v1.addPassenger("Nina") == True)
# can't add passengers past the max capacity
assert(v1.addPassenger("Beatrix") == False)
assert(str(v1) == "Paddleboat holds 2: ['Ken', 'Nina']")
# can remove the most recent passenger
assert(v1.removePassenger() == True)
assert(str(v1) == "Paddleboat holds 2: ['Ken']")
assert(v1.addPassenger("Beatrix") == True) # that frees up space!

v2 = Vehicle("Bus", 20)
# can't remove a passenger from an empty vehicle
assert(v2.removePassenger() == False)
assert(v2 == Vehicle("Bus", 20))
assert(v2 != v1)
assert(v2 != "Bus") # should not crash!

# a buggy is a vehicle with a name and an assumed capacity of 1
b1 = Buggy("SDC Buggy")
assert(b1.addPassenger("Driver") == True)
assert(str(b1) == "SDC Buggy holds 1: ['Driver']")
# a buggy knows whether it's currently being pushed
b1.startPushing()
# while it's being pushed, you can't remove passengers
# don't worry about adding passengers while pushing
assert(b1.removePassenger() == False)
b1.stopPushing()
# when the buggy is stopped, normal rules apply
assert(b1.removePassenger() == True)
assert(b1.removePassenger() == False)
```

Additional Space for Answer to Question 7

Additional Space for Answer to Question 7

8. (20 points) **Free Response: oddPerfectSquarePrefixes**

   Write a recursive function with backtracking oddPerfectSquarePrefixes(d) (or opsp(d) for short) that returns a number with d digits such that the number and every prefix of it is a perfect square, and the number is odd. For example, opsp(2) might return 49, since 4 and 49 are both perfect squares and 49 is odd. One possible return value for opsp(3) is 169 since 1, 16 and 169 are all perfect squares and 169 is odd. The function should return None if there is no number with d digits that follows this rule.

   Notes/Hints

   1. You must use recursion and backtracking to solve the problem to receive credit, even if another approach would work
   2. You may make use of wrapper functions and/or optional arguments.
   3. You should not return every possible solution, just the first one that you find.

Additional Space 1 for Answer to Question 8

Additional Space 1 for Answer to Question 8