15-112 Fall 2019 Midterm #2A
80 minutes

Name: ____________________________________

Andrew ID: ____________________@andrew.cmu.edu

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 (not scratch paper) to receive credit.**
- **If you use scratch paper, you must submit it with your andrew id on it, and we will ignore it.**
- **All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.**
- **You may use almostEqual() and roundHalfUp() without writing them. You must write everything else.**

Do not write below here

Question	**Points**	**Score**
1. Free Response #1	25	
2. Free Response #2	25	
3a. Fill-in-the-Blank	5	
3b. Free Response #3	25	
4. Short Answers	7	
5. BigOh	4	
6. Reasoning Over Code	4	
7. Code Tracing	5	
8a + 8b. CT Bonus	5 (bonus)	
TOTAL	100	

1. Free Response: DotsApp [25 pts]
Write an animation such that:

- You write the class Dot, and each dot is stored as an instance of that class.
- You write the class DotsApp that extends the App class.
- You follow all MVC conventions.
- Note: you can abbreviate self, app, canvas, and event as s, a, c, and e.

Your animation should have these features:

- A red dot of radius 20 is drawn centered in the canvas.
- On each mouse press, a new blue dot of radius 20 is drawn at that position.
- On each right-arrow key press, each blue dot moves 10 pixels to the right, but the red dot remains centered in the canvas.
- If the center of a blue dot moves at all off the canvas, the blue dot turns green and never moves again.
- 5 seconds after each blue dot is created, it is deleted if it has not yet turned green.

This page is blank (for your DotsApp solution, if needed).

2. Free Response: testVehicleAndSubclasses [25 pts]
Write the classes required to pass the following test code (without hardcoding any test cases).

Note: Your subclasses should contain at most a constructor and no other methods.

```
def testVehicleAndSubclasses():
    print('Testing Vehicle and Subclasses...', end='')
    v1 = Car('Honda', 2014)
    assert(isinstance(v1, Car) and
           isinstance(v1, Vehicle)) # all cars are vehicles!
    assert(str([v1]) == '[Honda car built in 2014]')
    v2 = Car('Honda', 2014)
    v3 = Car('Honda', 2019)
    v4 = Car('Chevy', 2014)
    assert((v1 == v2) and (v1 != v3) and (v1 != v4) and (v1 != 'Dont crash!'))
    v5 = Truck('Honda', 2014)
    assert(isinstance(v5, Truck) and
           isinstance(v5, Vehicle) and  # all trucks are vehicles
           not isinstance(v5, Car))     # but trucks are not cars
    assert(str(v5) == 'Honda truck built in 2014')
    assert(v5 != v1) # cars do not equal trucks!

    assert(Vehicle.counts == { 'car':4, 'truck':1 }) # this is a dictionary!
    v6 = Truck('Honda', 2014)
    assert(Vehicle.counts == { 'car':4, 'truck':2 })

    s = { v4, v6 }
    assert((v4 in s) and (v5 in s) and (v6 in s))
    assert((v1 not in s) and (v2 not in s) and (v3 not in s))
    print('Passed!')
```

This page is blank (for your testVehicleAndSubclasses solutions, if needed).

3a. Fill In The Blank: solveFromState [5 pts]
Complete the solveFromState method of the backtracking solver by filling in the blanks below with the LETTER corresponding to the correct code snippet below. There are more letters than blanks, so some will not be used! And some may be used twice! Remember, write the LETTER, not the whole code!

A: result != None
B: result == None
C: self.doMove(state, move)
D: self.solveFromState(childState)
E: self.checkConstraints
F: self.getLegalMoves(state)
G: self.stateSatisfiesConstraints(childState)
H: self.isSolutionState(state)
I: state
J: None

```
def solveFromState(self, state):
    if state in self.states:
        # we have already seen this state, so skip it

        return ________ #1
    self.states.add(state)

    if ________: #2
        # we found a solution, so return it!

        return ________ #3
    else:
        for move in ________: #4
            # 1. Apply the move

            childState = ________ #5
            # 2. Verify the move satisfies the backtracking constraints

            if ((________) or (not ________)): #6 and 7
                # 3. Add the move to our solution path
                self.moves.append(move)
                # 4. Try to recursively solve from this new state

                result = ________ #8
                # 5. If we solved it, then return the solution!

                if ________: #9
                    return result
                # 6. Else we did not solve it, so backtrack
                self.moves.pop()

        return ________ #10
```

3b. Free Response: HappyListSolver and HappyListState [25 pts]
Note: this is a backtracking problem. You must solve this by properly subclassing our generic backtracking solver classes. You do not have to write either BacktrackingPuzzleSolver or State, you can just subclass them.

Background: we will define a list L as "happy" (an invented term) if:

1. L is of length N and contains all the integers from 1 to N
2. For any two adjacent numbers in L, one of them is even and one is odd.
3. For any two adjacent numbers in L, their difference must be at least 2.

From this, we see that these lists are not happy:

- [1, 4, 1, 4] violates rule 1
- [1, 3, 4, 2] violates rule 2
- [1, 2, 3, 4] violates rule 3

And we see that these lists are happy:

- [1]
- [3, 6, 1, 4, 7, 2, 5]
- [1, 4, 7, 2, 5, 8, 3, 6]

With this in mind, write the classes HappyListSolver and HappyListState so that the following code works properly:

```
def findHappyList(n):
   # return a happy list of length n (or None if there isn't one)
   moves, solution = HappyListSolver(n).solve()
   return None if (solution == None) else solution.getList()

print(findHappyList(1)) # prints [1]
print(findHappyList(4)) # prints None
print(findHappyList(7)) # prints [3, 6, 1, 4, 7, 2, 5]
print(findHappyList(8)) # prints [1, 4, 7, 2, 5, 8, 3, 6]
```

We have outlined the functions, classes, and methods you need to write. You should fill in the 7 spaces provided. Don't forget to include the parameters you need in the __init__ methods!

```
class HappyListState(State):
   def __init__(self,                                          ):       # part 1 of 7
```

```
   def getList(self):                          # part 2 of 7
      # just return actual solution list so it can be printed
```

```
class HappyListSolver(BacktrackingPuzzleSolver):
   def __init__(self,                                          ):       # part 3 of 7
      # remember to set self.startState
```

```
   def isSolutionState(self, state):           # part 4 of 7
```

```
def getLegalMoves(self, state):           # part 5 of 7
```

```
def doMove(self, state, move):            # part 6 of 7
```

```
def stateSatisfiesConstraints(self, state):                     # part 7 of 7
```

4. **Short Answers** [7 pts]

1. Which of the following would produce an MVC Violation? Fill in the bubbles for all that apply

◯ A) Changing the model while in mousePressed.

◯ B) Changing the model while in redrawAll.

◯ C) Changing the model while in a helper function called by mousePressed.

◯ D) Changing the model while in a helper function called by redrawAll.

◯ E) None of these

2. Fill in the True or False bubble for each of the following:

◯ True	◯ False	Sets are unordered
◯ True	◯ False	Values in dictionaries must be unique
◯ True	◯ False	Keys in dictionaries must be unique
◯ True	◯ False	Hash functions must return unique integers so that hash(x) == hash(y) can only be true if x == y.

3. If we use mergesort on the list [5,2,8,4,6,1,3,7], what will the list be just before we start the final pass?

4. Which best describes a static method? (Choose one)

◯ A) A method that can only return the same static (unchanging) value.

◯ B) A method where we store the results in a dictionary and reuse the results to save time in future calls.

◯ C) A method that does not take 'self' as its first argument.

◯ D) There is no such thing as a static method in Python.

5. Say B is a subclass of A, and A has a method foo(s) that takes a string s and returns an int. Write the very short method foo(s) in B that overrides A's foo(s) so that it first converts s to uppercase and then works the same as A's foo.

6. Which of the following are listed in the course notes as a workaround instead of using mutable default values? Fill in the bubbles for all that apply

◯ A) Use a wrapper function.

◯ B) Use memoization.

◯ C) Use None as the default value.

◯ D) Do not mutate the default value.

◯ E) Use tail recursion

7. Memoization speeds up fib(n) dramatically. However, memoization would not speed up fact(n) very much. Which of the following best explains why this is true:

◯ A) In general, fib(n) is much larger than fact(n).

◯ B) In general, fib(n) requires many more duplicate recursive calls than fact(n).

◯ C) In general, fib(n) requires more steps per each recursive call than fact(n).

◯ D) In general, fib(n) uses more expensive arithmetic operations than fact(n).

5. Big-O runtimes [4 pts]
State the overall Big-O of each of these two functions in the boxes to the right

```
def bigOh1(L):
    # L is a list of N strings each of length M (or less)
    # You may assume each string only contains lowercase letters
    # Note that your answer may depend on both N and M
    d = dict()
    for s in L:
        for c in s:
            d[c] = d.get(c, 0) + 1
    result = ''
    for key in sorted(d.keys()):
        if (d[key] % 2 == 0):
            result += key
    return result
```

```
def bigOh2(L):
    # L is a list of integers, N = len(L)
    M = copy.copy(L)
    J = [ ]
    while (len(M) > 0):
        K = [ ]
        for value in L[0:len(L):5]:
            K.append(value + M[0])
        K.sort()
        J.append(K)
        M = M[:len(M)//2]
    return J
```

6. Reasoning Over Code [4 pts]
Find an argument for the following function that makes it return True. Place your answer (and nothing else) in the box beside the code:

```
def rc1(d):
    # hint: sum([]) == 0
    assert(isinstance(d, dict))
    for c in '15-112':
        if (c.isdigit()):
            i = min(int(c), 3)
            if ((len(d[i]) != i) or
                (0 in d[i]) or
                (sum(d[i]) != (2 + sum(d[i-1])))):
                return False
        elif (c not in d):
            return False
    return True
```

7. Code Tracing [5 pts]
Indicate what this prints. Place your answer (and nothing else) in the box.

```
def ct1(L):
    if (len(L) == 0):
        return [ ]
    else:
        return [max(L)] + ct1(L[1:-1]) + [min(L)]
print(ct1([1,2,3,2,1]))
```

8a. Bonus CT [2pts each]

Indicate what these print. Place your answers (and nothing else) in the box next to each block of code.

```
def bonusCt1(k):
    def f(n): return 0 if (n == 0) else 2*n-1+f(n-1)
    def g(n): return 0 if (n == 0) else 3*f(n)-3*n+1+g(n-1)
    n = f(g(k))
    while (g(n) < 10**6): n += 10
    return n

print(bonusCt1(2))
```

```
import math
def bonusCt2(z):
    def f(x): return 1 if (x == 0) else x*f(x-1)
    def g(x):
        q, s = 0, 1
        for i in range(1, 21, 2):
            q, s = q + s * x**i / f(i), -s
        return q
    b, bd = 0, 1
    for q in [z/1000 for z in range(1000)]:
        qd = abs(g(q) - z)
        if (qd < bd): b, bd = q, qd
    return 2**2 * b

print(bonusCt2(2**0.5/2))
```

8b. Bonus: Graphics Code Tracing [1 pt]
Sketch what this app will draw when run on a 400x400 canvas. Note: we are looking for a general pattern, not for pixel-perfect drawings, so just estimate positions and do not overly carefully measure them.

```
class GraphicsCTApp(App):
    def redrawAll(self, canvas):
        size = min(self.width, self.height)
        self.drawFractal(canvas, 0, 0, size, 2)

    def drawFractal(self, canvas, x0, y0, size, level):
        if (level == 0):
            canvas.create_text(x0+size/2, y0+size/2, text='X')
        else:
            for i in range(3):
                for j in range(3):
                    if (i+j > 2):
                        x1 = x0 + i*size//3
                        y1 = y0 + j*size//3
                        self.drawFractal(canvas, x1, y1, size//3, level-1)

GraphicsCTApp(width=400, height=400)
```

*this canvas is 400x400