15-112 Midterm #2 – version A – Spring 2015
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. Assume any imports are already included as required.**

DO NOT WRITE IN THIS AREA		
Part 1 (CT)	10 points	
Part 2 (RC)	5 points	
Part 3 (SA)	10 points	
Part 4 (FR)	15 points	
Part 5 (FR)	15 points	
Part 6 (FR)	15 points	
Part 7 (FR)	15 points	
Part 8 (FR)	15 points	
Part 9/bonus	5 points bonus	
Total	100 points	

1. **[10 pts] Code Tracing. Indicate what each will print:**

Statement(s):	**Prints:**

`def ct1(a, x):`
`  if (a == [ ]): return [ ]`
`  else: return [a[x%10]] + ct1(a[1:], x/10)`
 	
`print ct1(range(2,7), 1023)`
`# prints a list of length 5`
	_____

`def ct2(a):`
`  (x, result) = (0, [ ])`
`  while (sum(result) < 10):`
`  try:`
`  result.append(a[x])`
`  x += 5`
`  except:`
`  result.append(1)`
`  x /= 2`
`  return result`
`print ct2(range(2,6))`
`# prints a list of length 5`
	_____

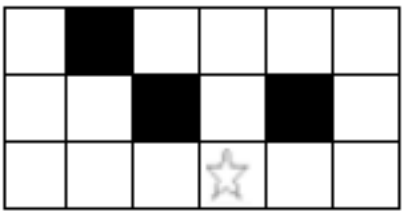
2. **[5 pts] Reasoning about code**
Find a value of x so the function rc(x) will return True. Place your answer in the box on the right.

`def rc(x):`
`  def f(x, depth=0):`
`  if (x == 0): return 10**depth`
`  else: return x%10 + f(x/10, depth+1)`
`  return (f(x) == 117)`
	x = _____

3. **[10 pts] Short Answer:** Answer each of the following very briefly.

A. If a Level-0 Sierpinski triangle is simply a black triangle, draw a Level-2 Sierpinski triangle, and circle all the Level-1 Sierpinski triangles inside of it.

B. Say we start with a 3x6 board and fill the black squares as shown below (in the code from our notes, the black squares would be green, and the white squares would be cyan). Then we right-click where the star is to start a floodFill from there. Write the numeric labels that result in each cell after the floodFill completes.



Hint #1: the numbers are the *depth* at each cell, so some labels may occur more than once.
Hint #2: the first label is 0, not 1 (so a 0 should be where the star is).
Hint #3: Our floodFill code recursively tries to go up, then down, then left, then right

C. **Fill-in-the-blanks: memoized recursive fib**

```
memoizedFibValues = ________________________    # 1 of 2

def fib(n):
    args = (n,)
    if (args in memoizedFibValues):
        return memoizedFibValues[args]
    if (n < 2):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)

    ___________________________________________  # 2 of 2
    return result
```

D. **Fill-in-the-blanks: recursive mergesort and merge**

```
def merge(A, B):
    if ((len(A) == 0) or (len(B) == 0)):
        return A+B
    else:
        if (A[0] < B[0]):
            return [A[0]] +
merge(________________________________)  # 1 of 5
        else:
            return [B[0]] +
merge(________________________________)  # 2 of 5

def mergesort(L):
    if (len(L) < 2):
        return ________________________________________   # 3 of 5
    else:
        mid = len(L)/2

        left = ________________________________________  # 4 of 5

        right = ________________________________________  # 5 of 5
        return merge(left, right)
```

E. Fill-in-the-blanks: recursive listFiles

```
def listFiles(path):
    if (os.path.isdir(path) == False):

        return ________________________________________  # 1 of 2
    else:

        files = [ ]
        for filename in os.listdir(path):

            ____________________________________________ # 2 of 2
        return files
```

4. **[15 pts] Free Response: hanoiLongHops(n)**
   In solving the Towers of Hanoi, sometimes we move a disk to a neighboring tower (which we will call a short hop), and sometimes we move a disk to a tower that is further away (a long hop). With this in mind, write the function hanoiLongHops(n), which you may abbreviate as hlh(n), that takes a positive int n, and returns the number of long-hop moves that our standard Hanoi solution takes when moving n disks from Tower 0 to Tower 1. For full credit, do not use any lists in your solution. For half-credit, you may use lists.

5. **[15 pts] Free Response: Playlist and Song classes**

Write the Playlist and Song classes so the following test code works. You may not use the Struct class from the course notes, and you may not hardcode the specific test cases.

```
song1 = Song("Happy", 10)
assert((song1.title == "Happy") and (song1.rating == 10))
song2 = Song("Joyful")
assert((song2.title == "Joyful") and (song2.rating == 6))
playlist1 = Playlist([song1, song2])
assert(playlist1.songs == [song1, song2])
assert(playlist1.averageRating() == 8) # average of 10 and 6
playlist2 = Playlist([song1])
assert(playlist2.songs == [song1])
assert(playlist2.averageRating() == 10)
assert(not (playlist1 == playlist2))
playlist2.addSong(song2)
assert(playlist2.songs == [song1, song2])
assert(playlist2.averageRating() == 8)
assert(playlist1 == playlist2)
```

[This page is blank (for PlayList and Song classes)]

6. **[15 pts] Free Response: BlobGame and Blob classes**

Background: we will say that a "blob" is a simple shape that can either be a circle or a square. It has an (x,y) location, a shape (circle or square), and a color. With this in mind, write two classes, BlobGame (that extends our eventBasedAnimation.Animation class) and Blob (that extends object), so that BlobGame().run() plays a simple game as such:

- Initially, there are 20 blobs, positioned randomly on the canvas. Their centers can be anywhere except within 10 pixels of either edge of the canvas, and their radiuses should be between 20 and 40. They should have a random color from among pink, yellow, or cyan. And they should have a random shape from among circle or square.
- If you click in a blob, it changes shape (from circle to square, or vice versa). Clicks outside of blobs are ignored.
- If multiple blobs are stacked on top of each other, and your click happens to intersect multiple blobs, then each one changes shape.
- Your test for a click in a blob must be different for circle blobs and square blobs.
- If the user presses 'r', the game resets with 20 new random blobs.
- You may not write any functions outside of classes. You must only write methods.

[This page is blank (for BlobGame and Blob classes)]

7. **[15 pts] Free Response: findRTP(digits)**

Background: A number n is a right-truncatable prime, or RTP, if every prefix of n (including n itself) are all prime. So, 593 is an RTP because 5, 59, and 593 are all prime.

With this in mind, write the function findRTP(digits) that takes a positive int, digits, and returns the smallest RTP with that many digits, or None if no such number exists.

To do this, you must use backtracking. At each step, try to add one more digit to the right of the number. Also, make sure you are only creating RTP's as you go.

You may assume that isPrime(n) is already written for you. While not required, you may find it is helpful to use an optional parameter, prefix, which is 0 by default and which holds the number that has been constructed so far.

Note: even though findRTP(8) returns 23399339, it runs almost instantly because backtracking rules out most numbers without trying them, so it actually calls isPrime very few times.

8. **[15 pts] Free Response: loopyOdds(sides, score)**

Background: in the game of Loopy (which we just invented), a player starts their turn by rolling two dice. They can keep rolling again until the sum of the dice is larger than the previous roll. Their score for that round is the number of times they rolled. So if they roll (5,3), then (4,4), they can keep rolling since 4+4 <= 5+3. Then they roll (1,3), and keep rolling. Then they roll (2,5) and stop, since 2+5 > 1+3. In total for that turn they rolled (5,3), (4,4), (1,3), (2,5), which is 4 rolls, so they score 4 points on that turn.

With this in mind, write the function, loopyOdds(sides, score), that takes two parameters -- a positive number of sides on each of the dice, and a possibly-negative int score -- and, using Monte Carlo techniques with 100,000 trials, returns the probability (as a value between 0.0 and 1.0) of getting exactly the given score in one turn of the Loopy game using dice with the given number of sides each. You must use Monte Carlo here, even if you know of a closed-form math solution.

[most of this page is blank]

Bonus/Optional: **[2.5 pts] What will this print?**

```
def bonus1():
    def f(x): return 4 if (x < -4) else 4 + f(x/4-5)**2
    def g(x, y=f(4)): return x if (f(x) > y) else g(x+1, y)
    return g(f(-123456789))
print bonus1()
```

Bonus/Optional: **[2.5 pts] What will this print?**

```
def bonus2(L, a = range(3)):
    while L: b = L.pop(); a.insert(-b%len(a), b)
    def f(a): return 0 if (a==[]) else (len(a)%2)*a[0] + f(a[1:])
    return f(a)
print bonus2(range(6,9))
```