15-112 Midterm #2 – Fall 2014
80 minutes

Name: _____

Andrew ID: _____@andrew.cmu.edu

Section: _____

INSTRUCTIONS

- **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!**

DO NOT WRITE IN THIS AREA		
Part 1 (SA/Classes)	12 points	
Part 2 (CT/FFill)	5 points	
Part 3 (CT/Fractal)	5 points	
Part 4 (SA)	15 points	
Part 5 (CT)	18 points	
Part 6 (FR)	15 points	
Part 7 (FR)	15 points	
Part 8 (FR)	15 points	
Part 9/bonus	6 points bonus	
Total	100 points	

1. **Short Answers: Classes** [12 pts; 4 pts each]

The following questions concern this code:

```
class A(object):
    def __init__(self, b, c):
        assert((type(b) == int) and (type(c) == int))
        self.d = b + c
```

1) Write one method to make this True:

```
((A(1, 2) == A(1, 2)) and
 (not (A(1, 2) == A(3, 4))) and
 (not (A(1, 2) == 3)))
```

2) Assuming == works properly, Write one method to make this True:

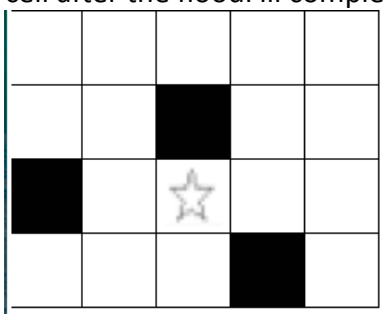
```
((A(1, 2) + A(3, 4) == A(3, 7)) and (A(1, 2) + 42 == A(3, 42)))
```

3) Write one class to make this True, with this restriction: your class may contain only one method (the constructor), and that method may not directly set any instance variables:

```
(isinstance(B(), A) and (B() == A(1, 2)))
```

2. **FloodFill Tracing** [5 pts]

Say we start with a 4x5 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: Here is an excerpt of the floodFill code:

```
def floodFill(row, col, color, depth=0):
        …
        floodFill(row-1, col, color, depth+1)
        floodFill(row+1, col, color, depth+1)
        floodFill(row, col-1, color, depth+1)
        floodFill(row, col+1, color, depth+1)
```

3. **Fractal Tracing** [5 pts]

Consider the following code:

```
def drawFractal(canvas, x, y, size, level):
    # (x,y) is the lower-left corner of the fractal
    if (level == 0):
        canvas.create_oval(x, y-size, x+size, y)
    else:
        drawFractal(canvas, x, y, size/4, level-1)
        drawFractal(canvas, x+size/4, y-size/4, size/4, level-1)
    # The next line draws a box around each level of the fractal
    canvas.create_rectangle(x, y-size, x+size, y)
```

At level 0, this draws a circle just inside a large square. Draw a picture of what this draws at level 2.

4. **Short Answers** [15 pts; 3 pts each]
Be very brief.

A. State how you can easily determine whether a method is static or not.

B. Say we have a subclass of EventBasedAnimationClass which overrides the run method, and after calling the super's run, it then tries to bind mouse motion events. This will not work as expected. Briefly state why not.

C. Give an example of some Python code that uses a closure.

D. Where in our Tetris code did we use a 3d list?

E. Rewrite the following so that f(x) works the same, but your code does not use any iteration. Assume that g is already written, and that x is a non-negative int.

```
def f(x):
    for i in xrange(x**2):
        print g(i)
```

5. **Code Tracing.** [18 pts; 3 pts each]
Indicate what each will print or draw:

Statement(s):	**Prints or Draws:**

`class G(object):`
`  def __init__(self, z=42):`
`  self.z = z`
`  def __str__(self):`
`  return "G(%d)" % self.z`
`  def foo(self):`
`  return (str(self),`
`  int(isinstance(self, H)),`
`  int(type(self)==H))`
`class H(G):`
`  def __str__(self):`
`  return "H(%d)" % self.z`
`class I(H):`
`  def bar(self): return "wow!"`
`for obj in [G(), H(5), I(10)]:`
`  print obj.foo()`	

`def f(x, y=14):`
`  print (x, y), # don't miss this!`
`  if (x > y): return x+y`
`  else: return f(2*x, y-1)`
`print f(3)`	

`def g(t="ab"):`
`  print t,  # don't miss this!`
`  try: t = string.ascii_uppercase[0:t]`
`  except: pass`
`  if (t == ""): return t`
`  else: return g(t[1:]) + t[0]`
`print g() + g(3)`	

Code Tracing continued.

Statement(s):	Prints or Draws:

`def h(x, y, depth=0):`
`    # This uses depth-based indentation`
`    # as in the course notes`
`    print "  "*depth, "h(%d,%d)" % (x,y)`
`    if (x+y <= 5):`
`        result = x+y`
`    else:`
`        result = h(x-2, y-2, depth+1)`
`        result += 2*h(x/2, y/2, depth+1)`
`    print "  "*depth, "-->", result`
`    return result`
`print h(4,6)`	

`def t1(x):`
`    def f(y): return lambda z: 10*y+z`
`    a = [f(y) for y in xrange(1,10,4)]`
`    b = [g(x) for g in a]`
`    return b`
`print t1(3)`	

`def t2(*args):`
`    if (len(args) == 0):`
`        return [ ]`
`    else:`
`        return ([args[0]] +`
`                t2(*args[-1:0:-1]))`
`print t2(3,4,5,6,7,8,9)`	

6. Free Response: hanoiMoves(n) [15 pts]

Write the function hanoiMoves(n) that takes a non-negative int n and returns a list of the moves required to solve the Towers of Hanoi with n discs. Assume the discs start on tower 0 and move to tower 1. Represent moves as a tuple (fromTower, toTower). Here is a test function for you:

```
def testHanoiMoves():
    print "Testing hanoiMoves()...",
    assert(hanoiMoves(1) == [(0,1)])
    assert(hanoiMoves(2) == [(0, 2), (0, 1), (2, 1)])
    assert(hanoiMoves(3) == [(0, 1), (0, 2), (1, 2),
                             (0, 1),
                             (2, 0), (2, 1), (0, 1)])
    print "Passed!"
```

7. **Free Response: TimeSinceLastMouseMotion** [15 pts]
Write the class TimeSinceLastMouseMotion that properly extends our BasicAnimationEventClass so that running it displays a canvas with a single number centered in the middle of it – the time, rounded to the nearest 1/10th of a second, since the last mouse motion event. You must not override the run method, so you must properly bind mouse motion events somewhere else. Also, each time the user moves the mouse, this time should reset to 0.0. You will want to use time.time(), which returns a float value of the time in seconds since some date a long time ago (so you will really be interested in differences in consecutive calls to time.time()).

8. **Free Response: splitBigFiles(path)** [15 pts]

Write the function splitBigFiles(path) that takes a path to a folder and finds every file in the folder or its subfolders that contains more than 1,000 characters. You may assume all the files are text files. Your function should replace each file that is larger than 1,000 characters with several files, splitting the original contents into 1,000-character-or-less blocks. These new files are named with the original file name followed by a number (so if file "foo.txt" contained 2500 characters, you would replace "foo.txt" with "foo.txt1" (the first 1,000 characters), "foo.txt2" (the next 1,000 characters), and "foo.txt3" (the last 500 characters). Your function should return True if it made any changes, and False otherwise. You may need os.remove(path) and any other os functions from the course notes. You also may use anything from the File IO notes. You may use loops so long as you use recursion in a meaningful way.

[this space intentionally left (mostly) blank, in case you need more room]

Bonus/Optional: **[3 pts] What will this print?**

```
def f(a=[1,2], b=[3,4]):
    if (len(a) == 1): return (a+b)
    else:
        a[0] += 1
        b.append(1)
        return f(a[:-1], b[1:])
print f(), f([5,6]), f()
```

Bonus/Optional: **[3 pts] What will this print?**

```
a = 0
def b(c): global a; a += 1; return a+c
d = lambda e: lambda f: b(10*e+f)
g = [d(h) for h in xrange(3)]
print [g[i](i) for i in xrange(len(g))]
```