CS 1301 – Spring 2009 Exam 3

Your Name: _____KEY_____
Your TA's Name: _____

Problem	Earned	Possible
1. Vocabulary		15
2. Mystery Drawing		5
3. My Picture		8
4. Blast Off		9
5. Break Them Out		14
6. Space The Groups		20
7. Smarter Squaring		10
8. Stock Games		10
9. Know your Sequence		9
Extra Credit		(3)
TOTAL:		100 (103 w/ ec)

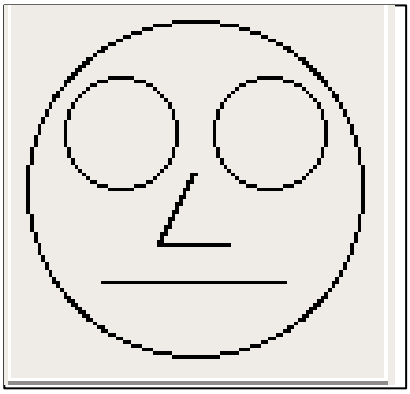
1. Vocabulary Matching (15 points). Write the number of the correct definition from the right column before each word in the left column.

__7__ Handle __1__ Pixel _13___ Local Variables __3__ Format Operator __5__ Immutable __15__ Short Circuit Evaluation __14__ Lambda __12__ Global Variables _11___ Semantic errors __2__ File __6__ Runtime errors __10__ Exception __8__ Raise __4__ Module __9__ Syntax Errors	1. The smallest distinct point in a graphic image. 2. A named entity, usually stored on a hard drive, floppy disk, or CD-ROM, that contains a stream of characters. 3. The `%` operator takes a format string and a tuple of expressions and yields a string that includes the expressions, formatted according to the format string. 4. A file containing definitions and statements intended to be imported by other programs. 5. A data type in which the elements can not be modified. 6. An error that occurs at runtime. 7. To prevent an exception from terminating a program using the `try` and `except` statements. 8. Statement used to signal an exception. 9. Produced by Python when it encounters a problem interpreting code. 10. Raised by the runtime system if something goes wrong while the program is running. 11. Problems with a program that compiles and runs but doesn't do the right thing. Example: An expression may not be evaluated in the order you expect, yielding an unexpected result. 12. Can be seen throughout a program module, even inside of functions. 13. Names defined within a function, are only visible within that function. 14. A block of code which can be executed as if it were a function but without a name. 15. When a boolean expression is evaluated the evaluation starts at the left hand expression and proceeds to the right, stopping when it is no longer necessary to evaluate any further to determine the final outcome.

2. Read Code: Mystery Drawing (5 Points)

Sketch the output of this code in the provided box (exact precision is not needed, but items should be positioned relatively correctly):

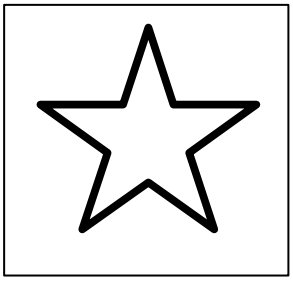
```
from myro import *
win = GraphWin("MyWin", 100, 100)
circle = Circle(Point(50, 50), 45)
circle2 = Circle(Point(30, 35), 15)
circle3 = Circle(Point(70, 35), 15)
line1 = Line(Point(25, 75), Point(75, 75))
line2 = Line(Point(50, 45), Point(40, 65))
line3 = Line(Point(40, 65), Point(60, 65))

circle.draw(win)
circle2.draw(win)
circle3.draw(win)
line1.draw(win)
line2.draw(win)
line3.draw(win)
```

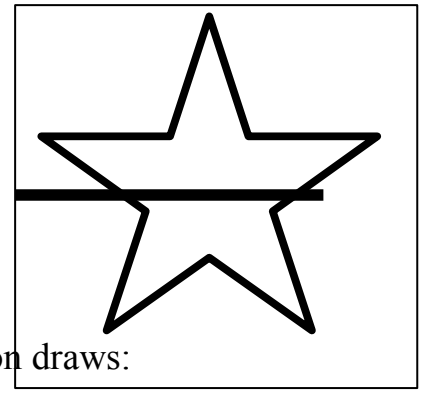


Grading: 1 point per facial feature, if they invert the whole thing, -2.

3. Read Code: My Picture (8 points)

In the following code, myPicture.jpg contains the following image and it's dimensions are 200x200 pixels:





Read the following code and sketch what the show() function draws:

```
p = loadPicture("myPicture.jpg")
for x in range( getWidth(p) - 50):
        for y in range( getHeight(p) / 2 ):
                pix = getPixel(p, x, y)
        setRed(pix, 0)
        setGreen(pix, 0)
        setBlue(pix, 0)
show(p)
```

Grading: 3 points if they draw a square, 6 points if they draw a line (even if white), 7 points if the line is black and horizontal, and full credit (8 points) if it stops ¾ of the way across.

4. Write Code – BlastOff (9 points)

Write three functions (blastOffWhile, blastOffFor, and blastOffRec). Each function accepts a single parameter which will be a positive integer and prints out a countdown like the following, where N is the number passed into the function:

```
N
.
.
.
3
2
1
Blastoff!
```

a. blastOffWhile must use only a while loop (3 pts):

```
def blastOffWhile(n):
    while (n > 0):
        print n
        n = n-1
    print "Blastoff!"
```

b. blastOffFor must use only a for loop (3 pts)

```
def blastOffFor(n):
    for i in range(n,0,-1):
        print i
    print "Blastoff!"
```

c. blastOffRec uses only recursion (no for or while loops!) (3pts):

```
def blastOffRec(n):
    if n > 0:
        print n
        blastOffRec(n-1)
    else:
        print "Blastoff!"
```

Grading: 3 points if it works correctly, 1 point if they miss a -1 or are off by one, 0 points for anything else.

5. Write Code! (14 points) Break Them Out

You have written some code to collect IR values as follows:

```
aList = []
for x in range(5):
     aList = aList + [ getIR() ]
     DoSomeMovement()
```

This results in aList holding values such as the following:

```
aList = [ [1,0],  [0,0], [1,1], [0,1], [0,0], [1,1] ]
```

But now your professor only wants you to use the RIGHT ir sensor value (the 2nd element in each sublist).

a. (10 pts) Write a function unzip2nd(aList) that returns a list made up of only the 2nd part of each sublist. For example, if you used it on the example aList, it should return a list like this: [0,0,1,1,0,1]. You may NOT use the map function.

```
def unzip2nd( aList):
   newList = []
   for item in aList:
      newList.append( item[1])
   return(newList)
```

Grading:

2 points for the correct def, function name, parameter, etc.
2 points for creating a new list
5 points for extracting the 2nd element of each item in the original list
2 points for placing the extracted elements into the new list in the appropriate order.
3 points for returning the list of extracted elements.

b.(4 pts) Now, re-write the unzip2nd() function **without** using a for or while loop. We suggest you use the map function. You may write another function of your choosing to help you out, or use a lambda function.

```
def unzip2nd( aList):
    result = map( lambda a: a[1], aList)
    return(result)
```

```
def bo2nd(item):
    return( item[1] )

def unzip2nd( aList):
    result = map( bo2nd, aList)
    return(result)
```

Grading: 4 points if they used MAP correctly. -1 for minor syntax errors.

6. Write Code – Space the Groups (Reading from/ Writing to files) (20 points)

Write a function called spaceGroups(*inFile*, *outFile*) that takes in two string filenames as parameters. The *inFile* will contain a list of names of students, with each line containing one name. However, there's no spacing to this file. You know that every 3 students are in a group, so you want to make a new output file *outFile* that contains all the names, but after every 3rd person, insert a new line to create spaces between the groups. This function doesn't need to return anything.

So if the *inFile* has:	The *outFile* should have:
Sam Peter Chris Danny Trevor Melody Ami	Sam Peter Chris Danny Trevor Melody Ami

Example:

```
def spaceGroups(infile,outFile):
    f = open(inFile, "r")
    out = open(outFile, "w")
    names = f.readlines()
    f.close()

    count = 0
    for line in names:
        out.write(line)
        count = count + 1
        if (count == 3):
            out.write("\n")
            count = 0


    out.close()
```

Grading:

2 pts – correct def/header
2 pts – open infile correctly
2 pts – open outfile correctly
5 pts – reads entire infile (and stops reading)
5 pts – Correctly spaces every 3 lines
2 pts – closes both files
2pts – does not return anything.

Example misc penalties:
-1 – minor syntax
-4 – Never stopping the read of infile (endless loop)

7. Write code: (10 points) Smarter Squaring

Write a function called **smarterSquaring** that takes in no parameters. Your function should prompt the user to enter some input ("Enter a number to square:") using the raw_input function. Your smarterSquaring function must make sure that what the user has entered can be converted to a float (e.g. "thirty point five" would be invalid but "35" and "3.5" would work.) If the input is invalid, the user should be prompted for input again, otherwise the function should return the square of the number that was entered. (For example, if the user typed 4.0, your function should return 16.0). You MUST use a try-except in this function.

```
def smarterSquaring():
    ui = raw_input("Enter a number to square:")
    try:
        Num = float(ui)
        return (Num * Num)
    except:
        print "Invalid number, try again!"
        return smarterSquaring()
```

Grading:

1 point– correct def/header

2 points – correctly prompts user (either separate print stmnt, or using raw_input)

1 points – uses float() to try and convert the string to a float

2 points – Correctly uses try/except to check if the float conversion works.

4 points – Handles float conversion exception, and keeps prompting for a new number until it works, then returns the squared number. It can either use looping or recursion for this.

8. Computational Complexity: Stock Games (10 points):

You are hired by a BigWinner Inc. to finish their stock recommendation software package after the previous developer was hit by a bus. The previous developer has left you two functions (RateStocksA, and RateStocksB) which are used to predict how much profit stocks will give in the next day. You test out each function with 1 stock, 2 stocks, and 5 stocks and find the following run-time and prediction accuracy results:

Number of Stocks	RateStocksA	RateStocksB
1	5 seconds / 89 % accuracy	1 second / 92% accuracy
2	10 seconds / 90 % accuracy	4 seconds / 92.5 % accuracy
5	25 seconds / 89.5 % accuracy	25 seconds / 91.9 % accuracy

a. What is the Big O (Computational Complexity) class of each function (N = Number of Stocks) (6 points)?

RateStocksA _____O(N)_____ RateStocksB____O(N^2)_____

Grading: 3 points each for correct answer.

b. Assuming you want to analyze the top 2500 stocks on the NYSE at the end of one trading day and decide what to purchase by the start of the next day to maximize your profit, which algorithm would you use? WHY? (4 points)

Example answer: The RateStocksB algorithm has a computational complexity of O(N^2) and based upon the time it took for 1, 2 and 5 stocks, it would take 6250000 seconds, or 1736 hours, or 72 days to finish analyzing 2500 stocks! Since it would not get done overnight, or even over a weekend, you must use the RateStockA algorithm, which will complete in approximately 12500 seconds, or 3.5 hours.

Grading: 1 point for the correct answer (RateStocksA) and 3 points for an explanation that explains that the RateStocksB algorithm would take too long.

9. Know your sequences! (9 points)

Three of the compound data types you have learned about are sequences. Name these three different types of sequences, give an example of each, and **state why they are different** from each other.

1. String – Only holds characters, immutable
2. Tuple – Can hold any type of data, immuatable
3. Lists – Can hold any type of data, mutable (can change elements)

Grading: 1 point for each name, 2 points for the descriptions.

Extra Credit: (3 points)

Write a BlastOffFunc method (as described in the BlastOff test problem) that uses elements of functional programming. It may NOT use a for or while loop, or recursion. You may create a helper function and call map, filter or reduce.

```
def printNum(N):
    print N

def blastOffFunc(n):
    map( printNum, range(n,0,-1) )
    print "Blastoff!"
```

Grading: 3 points if it works correctly, 1 point if they miss a -1 or are off by one, 0 points for anything else.