CMPSCI 119
Fall 2017
Monday, October 10, 2017
Midterm #1
Solution Key
Professor William T. Verts

<1> 20 Points – Do any 20; do more for extra credit. Correct answers are worth +1 point, blank answers are worth 0 points, but wrong answers are worth a –½ point penalty; if you don't know an answer, leaving it blank is usually better than a bad guess. The following statements have all been executed:

```
Frog = 15
Toad = 3.0
Goat = "COMPUTER SCIENCE ROCKS"
Newt = [2, 7.5, 3L, "Frog", [2,5,8], True]
Bird = (7, 3, 2, "Toad", 6.0)
```

Show the computed result for each problem; all are independent of one another. Indicate where a computation fails because of some form of error. Be careful about the *type* of the result, particularly int, float, long, bool, and complex types, and put proper quotes around string results, square brackets around lists, and parentheses around tuples.

	Question	Answer	
1.	`Frog + 1`	`16`	`(int)`
2.	`Toad + 1`	`4.0`	`(float)`
3.	`Frog / 2`	`7`	`(int)`
4.	`Toad / 2`	`1.5`	`(float)`
5.	`2 * Frog + 1`	`31`	`(int)`
6.	`len(Goat)`	`22`	`(int)`
7.	`len(Frog)`	ERROR	
8.	`len(Newt)`	`6`	`(int)`
9.	`len(Newt[3])`	`4`	`(int)`
10.	`Newt[4] + Toad`	ERROR	
11.	`Newt[4] + [Toad]`	`[2,5,8,3.0]`	`(list)`
12.	`Newt[2] + 1`	`4L`	`(long)`
13.	`Bird[3] + "s"`	`"Toads"`	`(string)`
14.	`Bird + Newt`	ERROR	
15.	`5 + Newt[-1]`	`6`	`(int)`
16.	`5 + (Toad < 1.0)`	`5`	`(int)`
17.	`range(Bird[0])`	`[0,1,2,3,4,5,6]`	`(list)`
18.	`range(1,Frog,4)`	`[1,5,9,13]`	`(list)`
19.	`range(Newt[0],-1,-1)`	`[2,1,0]`	`(list)`
20.	`range(Frog,len(Goat))`	`[15,16,17,18,19,20,21]`	
21.	`[0 for I in [2,8,1]]`	`[0,0,0]`	`(list)`
22.	`[I for I in [2,8,1]]`	`[2,8,1]`	`(list)`
23.	`[I for I in Newt[4]]`	`[2,5,8]`	`(list)`
24.	`[I*I for I in range(5)]`	`[0,1,4,9,16]`	`(list)`
25.	`[2*I+1 for I in range(3)]`	`[1,3,5]`	`(list)`

<2> 10 points – Show what is printed by the following code fragment for each given case:
(2 points each question, all or nothing)

```
if (N < 15):
    if (N < 6):
        print "A"
    else:
        print "B"
elif (N < 35):
    print "C"
else:
    print "D"
```

Case #1: **N=10**	**B**
Case #2: **N=5**	**A**
Case #3: **N=20**	**C**
Case #4: **N=40**	**D**
Case #5: **N=15**	**C**

<3> 15 Points – I have a list **L** containing a bunch of strings (such as **["Frog","Toad","Goat",…,"Bird","Fred"]**. You don't know how many items are in the list. Write a **while**-loop, using **I** as the loop control variable, that steps through and prints the items in the list in reverse order. Some framework code is provided for you.

```
EXPECTED:
    I = len(L)-1                    5 pts
    while (I >= 0):                 5 pts, Could be (I > -1)
        print L[I]                  3 pts
        I = I - 1                   2 pts, Could be I -= 1

ACCEPTABLE:
    I = 0                           2 pts
    while (I < len(L)):             5 pts
        print L[len(L)-I-1]         6 pts
        I = I + 1                   2 pts, Could be I += 1
```

Grading: For each line, remove points according to severity of error. For example, **(I > 0)** instead of **(I >= 0)** in the expected answer is -1 point.

<4> 20 Points – Show what is printed out as the result from calling **`Main()`** (four lines total):
(5 points each answer, all or nothing)

```
def Frog (M,J,Q):
    R = J - Q
    return M + R

def Toad(Z,Q,J):
    return Frog(Q,Z,J)

def Newt(R,Z,Q):
    return Toad(Q,Z+2,R)

def Main():
    print Frog(5,2,6)
    print Toad(1,6,3)
    print Newt(9,3,4)
    print Frog(2,6,2)
    return
```

Answer #1: 1

Answer #2: 4

Answer #3: 0

Answer #4: 6

<5> 15 Points – The code below contains syntax errors. Locate each one and indicate what the correction(s) should be. Don't rewrite any code statements; just correct the mistakes.

```
def Frog (P,Q):                         Missing :
    print P + Q
    Return                              Return → return

def Main():                             Missing ()
    Z == input("Enter a number --- '):  == → =, ' → "
    for I in rnage(10):                 rnage → range
        for J in [2,8,3,5]:
            if (I+J < Z):
                frog(I,J)               frog → Frog
      return                            shift indent left 2
```

I count 8 distinct errors. Assign +2 for each error found, -1 for each correct item misidentified as an error, but do not go above 15 nor below 0.

<6> 20 Points – The following code loads in a graphic picture from a file. Finish the function by doing the same process to each pixel **PX**, as follows: Set red to 255 if red was originally greater than 128 but set red to 0 if not; set green to 255 if green was originally greater than 128 but set green to 0 if not; set blue to 255 if blue was originally greater than 128 but set blue to 0 if not. **5 POINT BONUS QUESTION**: what is the maximum number of distinct colors that the resulting image could contain?

```
def Main():
    Filename = pickAFile()
    Canvas = makePicture(Filename)
    show(Canvas)
    for Y in range(getHeight(Canvas)):
        for X in range(getWidth(Canvas)):
            PX = getPixel(Canvas,X,Y)

            if getRed(PX) > 128:
                setRed(PX,255)
            else:
                setRed(PX,0)

            if getGreen(PX) > 128:
                setGreen(PX,255)
            else:
                setGreen(PX,0)

            if getBlue(PX) > 128:
                setBlue(PX,255)
            else:
                setBlue(PX,0)

        repaint(Canvas)
    return
```

Assign 5 points to each section, and the remaining 5 points for overall syntax issues. In each section, give full credit if the overall structure is basically correct: an `if-else` that tests a primary color and sets it appropriately. There are acceptable alternatives, such as putting the single statement of each body on the same line as the `if` or the `else`:

```
if getRed(PX) > 128: setRed(PX,255)
else: setRed(PX,0)
```

as well as purely computational solutions such as:

```
setRed(PX, getRed(PX) / 128 * 255)
```

BONUS QUESTION +5 POINTS: **8 distinct colors**