CMPSCI 119
Spring 2015
Final Exam
Friday, May 1, 2015
Solution Key

<1> 25 Points – Answer any **25** questions. Answer more for extra credit. Be careful about which variables and constants are ***integers***, which are ***floating-point numbers***, and which are ***strings***, ***lists***, ***tuples*** or ***dictionaries***! Indicate all floating-point answers with a decimal point, even if they are whole numbers, and indicate all string answers surrounded by quotes. If a statement contains a bug, answer **`error`**. For each question, assume variables have values as follows:

```
X = 1.5      Y = 5      L = [4, 7, 13, 2, 6]
S = "AArdvark"          T = (6, 0, 3)   D = {"Frog":1}
```

What value is returned by…

`3.0`	`X * 2`
`10`	`Y * 2`
`0.75`	`X / 2`
`2`	`Y / 2`
`0`	`int(X) / 2`
`0`	`int(X / 2)`
`2.5`	`float(Y) / 2`
`2.0`	`float(Y / 2)`
`4`	`L[0]`
`5`	`len(L)`
`error`	`len(L[1])`
`8`	`len(S)`
`18`	`Y + L[2]`
`error`	`S[10]`
`"k"`	`S[-1]`
`"dvark"`	`S[3:]`
`(6,0,3,6,0,3)`	`T + T`
`error`	`T + L`
`"AArdvarkAArdvark"`	`S + S`
`"Ak"`	`S[0] + S[7] + ""`
`[1.5,1.5,1.5]`	`[X, X, X]`
`[1.5,(6,0,3)]`	`[X, T]`
`6`	`D["Frog"] + 5`
`error`	`D["Toad"] + 5`
`[0,1,2,3,4]`	`range(Y)`
`[1,3]`	`range(1,Y,2)`
`[-1,0,1]`	`range(-1,2)`
`[0,1,2]`	`range(len(T))`
`[0] (or error)`	`range(X)`
`"AArdvarks"`	`S + "s"`
`error`	`float(S)`
`[6,7,8]`	`[Z+1 for Z in [5,6,7]]`
`[0,0,0,0,0]`	`[0 for Z in range(5)]`
`["A","A","r","d","v","a","r","k"]`	`[Z for Z in S]`
`[0,1,4,9]`	`[Z*Z for Z in range(4)]`

<2> 10 Points – More questions about Python. Answer all of these; no extra credit.

F	T/F: The last value of variable **I** in **for I in range(10):** is 10.
F	T/F: All **while** loops can be rewritten as **for** loops.
T	T/F: All **for** loops can be rewritten as **while** loops.
T	T/F: Proper indentation is critical to the execution of a Python program.
F	T/F: Python programs can run by themselves without an interpreter.
T	T/F: **makeColor(255,0,0)** and **red** are the same color.
T	T/F: **for px in getPixels(Canvas):** scans all pixels in an image.
==	How is the "is equal to" operator written in Python?
for **while**	What Python keywords are used to indicate loops?
""" **'''**	What symbol(s) are used so strings in Python can span multiple lines?

<3> 10 Points – We need a function that does not exist in Python. Write a complete new Python function called **Gap** with one numeric argument **N** to compute and return (but not print!) the value +10 if **N** is greater than zero, return -10 if **N** is less than zero, and return 0 if **N** is exactly zero, using only simple Python statements.

```
def Gap(N):
    if (N > 0): return +10
    if (N < 0): return -10
    return 0
```

-or-

```
def Gap(N):
    if (N > 0): Result = +10
    elif (N < 0): Result = -10
    else: Result = 0
    return Result
```

<4> 5 Points – Finish the function below to write out all integers from 0 to 100, and their square roots, to the external text file indicated by **Filename**.

```
def Lister(Filename):

    MyFile = open(Filename, "w")
    for I in range(101):
        MyFile.write(str(I) + ":" + str(math.sqrt(I)) + "\n")
    MyFile.close()

    return
```

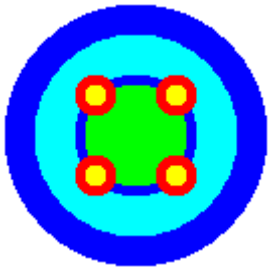
<5> 15 Points – The following code framework loads in and displays a graphics image. Complete the code to step through and process every pixel in the image as follows. First compute the pixel's brightness as the average of the red, green, and blue values. If the brightness is above 192 replace the R value with G, the G value with B, and the B value with R (rotate colors one direction). If the brightness is below 64 replace the R value with B, the B value with G, and the G value with R (rotate colors the other direction). If the brightness is in the range 64…192 leave the colors unchanged.

```
def Main():
    Filename = pickAFile()
    Canvas = makePicture(Filename)
    show(Canvas)
    for PX in getPixels(Canvas):

        R = getRed(PX)
        G = getGreen(PX)
        B = getBlue(PX)
        Brightness = (R + G + B) / 3

        if (Brightness > 192):
            Temp = R
            R = G
            G = B
            B = Temp

        elif (Brightness < 64):
            Temp = R
            R = B
            B = G
            G = Temp

        setRed(PX,R)
        setGreen(PX,G)
        setBlue(PX,B)

    repaint(Canvas)
    return
```

<6> 10 Points – The object below consists of a number of "spots" where a spot is one filled circle on top of another filled circle, as shown in the **addSpot** function. In **addSpot**, parameters **R1** and **C1** describe the radius and color of the first circle painted, while **R2** and **C2** describe the second circle painted. The object consists of six spots:

Spot #1: blue, radius 65, cyan, radius 50
Spot #2: blue, radius 30, green, radius 25
Spots #3, #4, #5, #6: red, radius 10, yellow, radius 5

Spots #3-#6 are offset from the center of the object by 20 in both the *x* and *y* directions.



Complete the **addFigure** function to paint one of these objects at coordinates **Xc**, **Yc**.

```
def INT(N):
    return int(round(N))

def addCircle (Canvas, Xc, Yc, R, NewColor=white):
    addOvalFilled(Canvas, Xc-R, Yc-R, 2*R+1, 2*R+1, NewColor)
    return

def addSpot (Canvas, Xc, Yc, R1, C1, R2, C2):
    addCircle (Canvas, Xc, Yc, R1, C1)
    addCircle (Canvas, Xc, Yc, R2, C2)
    return

def addFigure (Canvas, Xc, Yc):

    addSpot (Canvas, Xc   , Yc   , 65  , blue , 50  , cyan  )

    addSpot (Canvas, Xc   , Yc   , 30  , blue , 25  , green )

    addSpot (Canvas, Xc-20, Yc-20, 10  , red  , 5   , yellow)

    addSpot (Canvas, Xc+20, Yc-20, 10  , red  , 5   , yellow)

    addSpot (Canvas, Xc-20, Yc+20, 10  , red  , 5   , yellow)

    addSpot (Canvas, Xc+20, Yc+20, 10  , red  , 5   , yellow)

    return
```

<7> 5 Points – Short Answer – Function **addSpot** is outside **addFigure**. Can any other function in the program "see" the **addSpot** function? Could **addSpot** be nested inside **addFigure**?

Yes, any functions below **addFigure** could use **addSpot**.

Yes, **addSpot** would work OK if it was nested inside **addFigure**, but no other function could use it in that case.

<8> 15 Points – Assuming that functions **addFigure** exists and works as specified on the previous page, and that a blending function **BlendInt** has been written as shown below (**no** other blending functions are available). Complete the **Animate** function below to show the **addFigure** object moving across the screen from point **P0** to point **P1** (the number of frames in the animation is **Frames**), pausing for 0.1 seconds per frame:

```
import time

def BlendInt (P0,P1,T):
    return int((P1 – P0) * float(T) + P0)

def Animate(Canvas,Frames,P0,P1):

    for I in range(Frames):
        setAllPixelsToAColor(Canvas,white)
        T =float(I) / (Frames-1)
        X = BlendInt(P0[0], P1[0], T)
        Y = BlendInt(P0[1], P1[1], T)
        addFigure(Canvas, X, Y)
        repaint(Canvas)
        time.sleep(0.1)

    return

def Main():
    Canvas = makeEmptyPicture(1000,800)
    Animate(Canvas, 300, [100,175], [750,560])
    return
```

<9> 5 Points – Short Answer – Why do the 3D orthographic transforms that we use assume a fixed angle of 30 degrees? (You may answer on the back of this page.)

Because it looks cool! Empirically, 30° gives an appearance that, while not true perspective, looks close enough to be acceptable. Also, that specific angle has sine and cosine values which are easy to pre-compute without using the expensive-in-time `sin` and `cos` functions. That is, sin(30°) = ½ = 0.5, and cos(30°) = (√3)/2= 0.866…, which can be computed once at the beginning of the program.