CMPSCI 119
Fall 2018
Thursday, November 29, 2018
Midterm #3 Solution Key
Professor William T. Verts

<1> 25 Points – What is the value of each expression below? Answer any 25; answer more for extra credit. Answer "Error" if an expression cannot be computed for any reason. Scoring:

+1: Completely correct answers
+½: Incorrect data types, lists without square brackets, strings without quotes, etc.
0: Blank answers
–½: Incorrect answers (better to leave it blank than to guess)

```
Critters = ["Dog","Cat","Bird","Snake","Frog","Bat"]
Counter = 5
Result = True
Score = 3.5
Message = "Big String"
```

1.	`2`	`Counter / 2`
2.	`10`	`Counter * 2`
3.	`10.0`	`Counter * 2.0`
4.	`7.0`	`Score * 2`
5.	`6`	`Counter + Result`
6.	`8.5`	`Counter + Score`
7.	`6`	`len(Critters)`
8.	`ERROR`	`len(Counter)`
9.	`4`	`len(Critters[2])`
10.	`ERROR`	`Critters + "Toad"`
11.	`[5, True, 3.5]`	`[Counter, Result, Score]`
12.	`13.5`	`Score + 10L`
13.	`[0,1,2,3,4]`	`range(Counter)`
14.	`[0,1,2]`	`range(len(Critters[-1]))`
15.	`ERROR`	`range(Message)`
16.	`[3,4,5]`	`range(int(Score),len(Critters))`
17.	`[5,7,9,11,13]`	`range(Counter,15,2)`
18.	`[]`	`range(10,0)`
19.	`[1,0,-1]`	`range(1,-2,-1)`
20.	`["Cat","Bird","Snake"]`	`Critters[1:4]`
21.	`["Dog","Cat","Bird"]`	`Critters[:3]`
22.	`["Frog","Bat"]`	`Critters[4:]`
23.	`[0,0,0,0,0]`	`[0 for I in range(Counter)]`
24.	`[3,3,4,5,4,3]`	`[len(X) for X in Critters]`
25.	`["S","n","a","k","e"]`	`[CH for CH in Critters[3]]`
26.	`[6,3,2,9]`	`[Q+1 for Q in [5,2,1,8]]`
27.	`["D","o","g"]`	`[Z for Z in Critters[0]]`
28.	`["Dogs","Cats"]`	`[W+"s" for W in Critters[:2]]`
29.	`[]`	`[X+1 for X in range(0)]`
30.	`["B","i","g"]`	`[Message[I] for I in range(len(Message)) if I<3]`

<2> 20 Points – (2 points each answer) When **`Main()`** is called there will be exactly ten lines of output printed. What are they?

```
def FN1 (B,A,T=3):
    global Frog
    Frog = B + T - A
    print Frog + B
    return

def FN2 (R,A,T=1):
    global Frog
    print Frog
    FN1 (T,R+Frog,A)
    return

def Main():
    FN1 (1,2)
    FN2 (4,2,1)
    FN1 (1,2,1)
    FN2 (1,1)
    FN2 (1,1,3)
    FN2 (2,1,2)
    return
```

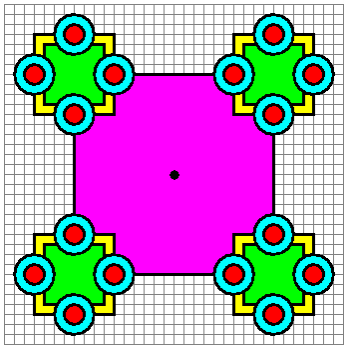
```
# Answer #1    3
#
# Answer #2    2
#
# Answer #3   -2
#
# Answer #4    1
#
# Answer #5    0
#
# Answer #6    2
#
# Answer #7    1
#
# Answer #8    5
#
# Answer #9    2
#
# Answer #10   1
#
```

<3> 10 Points – I have a program that scans every pixel in an image, and calls function **`Process`** with each pixel in variable **`PX`**. Complete the **`Process`** function to transform the pixel by converting it to red if the average of green and blue is greater than 128, and converting it to cyan otherwise. (The only JES functions you need are **`getRed`**, **`getGreen`**, **`getBlue`**, and **`setColor`**, but no canvases or loops are needed.)

```
def Process(PX):
    if (getGreen(PX)+getBlue(PX))/2 > 128:
        setColor(PX,red)
    else:
        setColor(PX,cyan)
    return
```

Remove 1 point per syntax error, remove 5 points for making up statements that are not Python, and remove 10 points for "solve problem through magic" solutions.

<4> 25 Points – A **Blodgett** is a magenta square of side radius 100, with a **Juntura** centered at each corner. A **Juntura** is a yellow square of radius 40, with a green square of side radius 30 on top, and a **Siletz** centered on each side, as shown. A **Siletz** is a cyan circle of radius 20, with a red circle of radius 10 on top. The **Circle** and **Square** functions are already provided.



Fill in the blanks in the functions below to complete the drawing of a **Blodgett** at location <X,Y> (which is the center of the **Blodgett**, shown with a dot).

```
def Circle (Canvas, Xc, Yc, Radius, NewColor=black):   …
def Square (Canvas, Xc, Yc, Radius, NewColor=black):   …

def Blodgett (Canvas,X,Y):

    def Siletz (X,Y):
        Circle(Canvas,__X__,__Y__,__20__,__cyan__)
        Circle(Canvas,__X__,__Y__,__10__,__red__)
        return

    def Juntura (X,Y):
        Square(Canvas,__X__,__Y__,__40__,__yellow__)
        Square(Canvas,__X__,__Y__,__30__,__green__)
        Siletz(__X__,__Y+40__)
        Siletz(__X__,__Y-40__)
        Siletz(__X-40__,__Y__)
        Siletz(__X+40__,__Y__)
        return

    Square(Canvas,__X__,__Y__,__100__,__magenta__)
    Juntura(__X+100__,__Y+100__)
    Juntura(__X-100__,__Y+100__)
    Juntura(__X+100__,__Y-100__)
    Juntura(__X-100__,__Y-100__)
    return
```

Remove one-half point per error in slots for each **Circle** and **Square** argument (20 total, for 10 points); remove one point per error in slots for each **Siletz** and **Juntura** argument (16 total, for 16 points), but do not go below zero. Calls to **Siletz** and calls to **Juntura** can be in any order.

<5> 5 Points – Refer to the **`BlendPoints`** function we developed in class (or see the bottom of page 298 of the Companion). What is returned by the following function call?

```
BlendPoints ([10,8,4], [16,-8], 0.5)
```

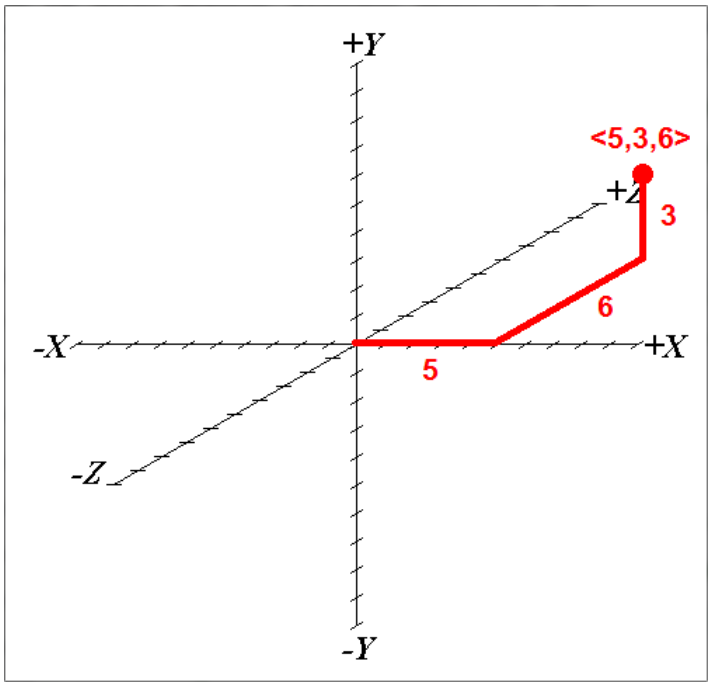
`[13,0]` if using the code developed in class, **error** if using the code on page 298. Accept either answer.

Subtract 1 point if they include a third value, as in `[13,0,2]`.

The code from class uses the length of the smallest list to control how many dimensions (times through the loop) will be used, so with a list of length 3 and a list of length 2 the result will be in two dimensions.

The code on page 298 uses the length of the first list only, and since the first list is longer than the second, Python will try to access the third element of a two-element list, causing an error crash.

<6> 5 Points – Show on the orthographic axes below where the point <5,3,6> is located. Refer to page 320 in the Companion for how the answer should be drawn.



<7> 10 Points – Write a complete Python function called **`Group`**, with one parameter **`N`**, that asks the user for **`N`** individual numbers (using the python **`input`** function) and returns the average of those numbers.

Here are two acceptable answers. Accept either approach, removing 1 point per syntax error for reasonable code, but remove 5 points if the students make up things like an "average" function or "get N numbers" or other things which are obviously not Python. Remove 10 points for essentially "solve problem by magic" answers.

```
def Group (N):
    T = 0.0
    for I in range(N): T = T + input("Enter a number --- ")
    return T / N
```

```
def Group (N):
    T = 0.0
    I = 1
    while (I <= N):
        X = input("Enter a number --- ")
        T = T + X
    Result = T / N
    return Result
```