CMPSCI 119
Spring 2017
Wednesday, March 29, 2017
Midterm #2
Professor William T. Verts

NAME	

QUESTION	POINTS	SCORE
1	25+5	
2	5	
3	24	
4	20	
5	16	
6	10	
TOTAL	100	

<1> 25 Points – What is the value of each expression below? Answer any 25; answer more for extra credit. Variable `S = "SPRINGER SPANIEL"`, `L = ["DOG", 6.9, 4]`, `X = 7`, `T = (4.3, "RABBIT", 5)`, and `D = {5:"S", 9:23.8, 1:"U"}`. Answer "Error" if an expression cannot be computed for any reason. Incorrect answers will be assessed as -1, correct answers as +1, and blank answers as 0. Your score will be the total (but will not go below zero). For example, if you answer all 30 problems but get 25 right and 5 wrong, your final score will be 20. (-½ for missing quotes or type errors.)

1.	`3` (int)	`X / 2`
2.	`8L` (long)	`X + 1L`
3.	`7.0` (float)	`X * 1.0`
4.	`13.9` (float)	`X + L[1]`
5.	`30.8` (float)	`X + D[9]`
6.	`ERROR` (int + tuple)	`X + T`
7.	`3`	`len(L)`
8.	`3`	`len(L[0])`
9.	`ERROR` (len(float))	`len(L[1])`
10.	`3`	`len(T)`
11.	`6`	`len(T) + len(L)`
12.	`3`	`len(D)`
13.	`ERROR` (tuple + list)	`T + L`
14.	`"DOGRABBIT"`	`L[0] + T[1]`
15.	`"L"`	`S[-1]`
16.	`ERROR` (out of range)	`S[len(S)]`
17.	`" "` (quoted blank)	`S[8]`
18.	`"RUG"`	`T[1][0] + D[1] + L[0][2]`
19.	`ERROR` (no key = 2)	`D[2]`
20.	`4` (int)	`int(T[0])`
21.	`23` (int)	`int(D[9])`
22.	`4.0` (float)	`round(T[0])`
23.	`24.0` (float)	`round(D[9])`
24.	`4.0` (float)	`round(L[2])`
25.	`ERROR` (round(string))	`round(T[1])`
26.	`[1,9,25]`	`[I*I for I in range(1,7,2)]`
27.	`["D","O","G"]`	`[Q for Q in L[0]]`
28.	`[0,1,2,3,4,5,6]`	`[Q for Q in range(X)]`
29.	`[0,0,0]`	`[0 for Z in [5,2,8]]`
30.	`[6,3,9]`	`[Z+1 for Z in [5,2,8]]`

<2> 5 Points – Based on the earlier variable definitions, which of the following expressions are legal, which are illegal, and why? (-1 per error, do not go below zero.)

`S[0] = "X"`	`L[0] = "X"`	`T[0] = "X"`
Illegal (immutable)	**Legal (mutable)**	**Illegal (immutable)**

<3> 24 Points – What is printed out when **`Main()`** is called: (4 points each)

```
def FN(W,Q,X=2):          |    def Main():          |
    print W+Q-X           |        A = 4            |
    return                |        B = 7            |
                          |        FN(A,B)          |
def F2(Q,Z,W=3):          |        FN(2,B,A)        |
    FN(Z,Q)               |        F2(A,A)          |
    print W+Z             |        F2(5,B,A)        |
    return                |        return           |
```

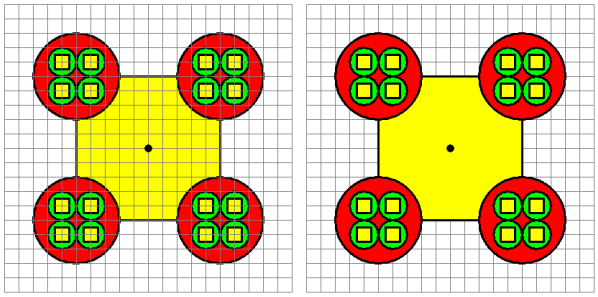
Answers:

1. 9
2. 5
3. 6
4. 7
5. 10
6. 11

<4> 20 Points – Write code inside the **`Weird`** function below to swap randomly selected pairs of pixels a million times. That is, your code picks coordinates <X1,Y1> and <X2,Y2> randomly, then swaps the colors of the corresponding pixels. You may assume that the statement **`import random`** has already been executed earlier in the program. (2 points per line.)

```
def Weird (Canvas):
    for I in range(1000000):

        X1 = random.randrange(getWidth(Canvas))

        Y1 = random.randrange(getHeight(Canvas))

        X2 = random.randrange(getWidth(Canvas))

        Y2 = random.randrange(getHeight(Canvas))

        PX1 = getPixel(Canvas, X1, Y1)

        PX2 = getPixel(Canvas, X2, Y2)

        C1  = getColor(PX1)

        C2  = getColor(PX2)

        setColor(PX1, C2)

        setColor(PX2, C1)

        if ((I % 100) == 0): repaint(Canvas)
    return
```

<5> 16 Points – (½ point per slot.) A **Whatzit** is a yellow square of radius 50 (the radius of a square is from center-to-side, not center-to-corner), with a **Blodge** at each corner. A **Blodge** is a red circle of radius 30; with a **Gronk** 10 pixels diagonally away from its center in all four directions. Each **Gronk** is a green circle of radius 10 with a yellow square of radius 5 on top of it. Fill in the blanks below to complete the drawing of a **Whatzit** centered at location <X,Y> (shown with a dot). The **addCircle** and **addSquare** functions are already provided.



```
def addCircle (Canvas, X, Y, Radius, NewColor=black):   …
def addSquare (Canvas, X, Y, Radius, NewColor=black):   …

def Whatzit (Canvas, X, Y):

    def Blodge (X, Y):

        def Gronk (X, Y):

            addCircle(Canvas,  ___X,  ___Y,  __10,  _green)

            addSquare(Canvas,  ___X,  ___Y,  ___5, yellow)

            return

        addCircle(Canvas,  ___X,  ___Y,  __30,  _red)

        Gronk (X-10, Y-10)          These four calls

        Gronk (X+10, Y-10)          can be in

        Gronk (X-10, Y+10)          any order.

        Gronk (X+10, Y+10)

        return

    addSquare(Canvas,  ___X,  ___Y,  __50, yellow)

    Blodge(X-50, Y-50)          These four calls

    Blodge(X+50, Y-50)          can be in

    Blodge(X-50, Y+50)          any order.

    Blodge(X+50, Y+50)

    return
```

<6> 10 Points – Examine the **Whatzit** program on the previous page, and assume that all of the blanks have been filled in correctly to draw the indicated figure. Answer the following questions:

A. How many individual calls to **addCircle** and **addSquare** would be required if the **Whatzit** function was not designed as a hierarchical decomposition? (That is, **Whatzit** only contains calls to **addCircle** and **addSquare**, and does not define either **Blodge** or **Gronk**.)

Calls to **addCircle**: **20** (1 point)

Calls to **addSquare**: **17** (1 point)

B. Can **Blodge** be called from a function outside of **Whatzit**?

NO. (2 points) (**Blodge** is defined locally inside **Whatzit**.)

C. Can **Gronk** be called by **Whatzit**?

NO. (2 points) (**Gronk** is defined locally inside **Blodge**.)

D. Why don't **Blodge** and **Gronk** need to be passed **Canvas** through their parameters?

Because **Canvas** is passed into **Whatzit**, and is locally "global" to **Blodge** and **Gronk**. (2 points)

E. Could **Gronk** be moved outside and above **Blodge** (but still remain inside **Whatzit**) and have everything still work?

YES. (2 points)