CMPSCI 119
Spring 2019
Friday, March 1, 2019
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 incur 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:

```
Orange = 15
Lemon  = 6.5
Lime   = "Avocado and Salsa"
Banana = [9, "Kiwi", 7.5, "Grape", [5, "9", 3], -12]
Apple  = ([1,2,3],[4,5,6],[7,8,9])
```

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 (either ' or ") around string results, square brackets around lists, and parentheses around tuples.

	Question	Answer	
1.	`Orange + Lemon`	`21.5`	`float`
2.	`Orange * 2`	`30`	`int`
3.	`Orange * 2.0`	`30.0`	`float`
4.	`Orange / 2`	`7.5`	`float`
5.	`Orange // 2`	`7`	`int`
6.	`Orange // 2.0`	`7.0`	`float`
7.	`Orange + Lime`	`ERROR`	`int+string`
8.	`len(Lemon)`	`ERROR`	`len(float)`
9.	`len(Lime)`	`17`	`int`
10.	`len(Banana)`	`6`	`int`
11.	`len(Apple)`	`3`	`int`
12.	`Lime[1]`	`"v"`	`string`
13.	`Banana[1]`	`"Kiwi"`	`string`
14.	`Apple[1]`	`[4,5,6]`	`list`
15.	`Banana[1][1]`	`"i"`	`string`
16.	`len(Banana[3])`	`5`	`int`
17.	`Banana[-1]`	`-12`	`int`
18.	`Banana[4] + Apple[2]`	`[5,"9",3,7,8,9]`	`list`
19.	`Banana[0] + Banana[4][1]`	`ERROR`	`int+string`
20.	`Banana[1] + Banana[4][1]`	`"Kiwi9"`	`string`
21.	`Banana + Apple`	`ERROR`	`list+tuple`
22.	`Apple[0] + Apple[1]`	`[1,2,3,4,5,6]`	`list`
23.	`range(Banana[0])`	`[0,1,2,3,4,5,6,7,8]`	`list`
24.	`range(Banana[-1],Orange,4)`	`[-12,-8,-4,0,4,8,12]`	`list`
25.	`range(Orange,3)`	`[]`	`list`

<2> 15 Points – Convert the **`while`**-loop below into a **`for`**-loop that does the same thing. Your solution needs at most <u>two</u> lines of Python code.

```
Count = 8
while (Count < 96):
    print (Count, math.sqrt(Count))
    Count = Count + 7
```

```
for Count in range(8,96,7):
    print (Count, math.sqrt(Count))
```

-or-

```
for Count in range(8,96,7): print (Count, math.sqrt(Count))
```

Remove points as follows:
-3 For including explicit **`Count = 8`** statement before `for`-loop.
-3 For including explicit **`Count = Count + 7`** statement inside `for`-loop.
-2 For omitting **`range`** function (i.e., **`for Count in (8,96,7):`**)
-1 Per syntax error (omitting **`:`** at end of `for`-loop, omitting parentheses on `print`, capitalization errors, etc.

Do not penalize for including **`import math`** at beginning (irrelevant code).

<3> 20 Points – Show what is printed out as the result from calling **`Main()`** (four lines total): 5 points per answer.

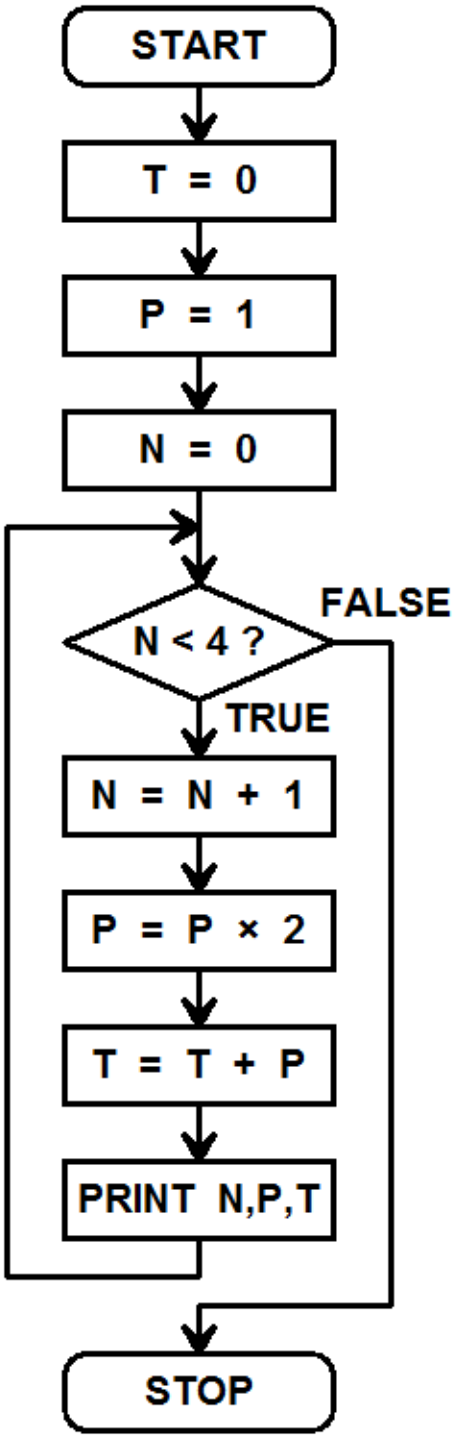
```
def F1(A,B,C=5):
    D = A + B * C
    return D + 1

def F2(B,D=1,A=4):
    return F1(D,A,B)

def F3(C,D,A):
    B = F1(C,A, F2(D))
    return B

def Main():
    print (F1(5,2,3))      # Answer #1:  12
    print (F2(4,8))        # Answer #2:  25
    print (F3(3,2,4))      # Answer #3:  44
    print (F1(6,-1))       # Answer #4:  2
    return
```

<4> 15 Points – Trace the following flowchart. Any time a variable value is updated, write the new value in the appropriate box and crossing out any previous value. There will be four lines printed as a result of running the program; show the values of the variables in each of those lines. 1 point per output value, 1 point per variable box.



Variables

N	~~0~~, ~~1~~, ~~2~~, ~~3~~, 4
P	~~1~~, ~~2~~, ~~4~~, ~~8~~, 16
T	~~0~~, ~~2~~, ~~6~~, ~~14~~, 30

OUTPUT FROM PRINTS

LINE #1: N= 1 P= 2 T= 2

LINE #2: N= 2 P= 4 T= 6

LINE #3: N= 3 P= 8 T= 14

LINE #4: N= 4 P= 16 T= 30

<5> 15 Points – Convert the flowchart on the previous page into the equivalent and correct Python 3 code. Your solution won't need any function definitions.

```
T = 0

P = 1

N = 0

while N < 4:

    N = N + 1

    P = P * 2

    T = T + P

    print (N,P,T)
```

Remove 1 point for each syntax error (indentation, omitting colons or parentheses, using the wrong symbol for multiplication, capitalization, etc.), but do not go below zero.

<6> 15 Points – Complete the following function to return **`True`** if **`N`** is greater than or equal to 0 and less than or equal to 100, but it must return **`False`** if **`N`** is either less than zero or greater than 100. Your solution must ***NOT*** contain any **`print`** statements.

Any of the following solutions are acceptable. There may be other valid approaches as well. Once a solution type has been identified, remove 1 point per error (using a **`print`**, omitting colons on **`if`**, **`elif`**, or **`else`** statements, switching **`<`**, **`<=`**, **`>`**, **`>=`** as appropriate, omitting parentheses on clauses using **`and`**, capitalization errors on **`True`** or **`False`**, etc.)

```
def MyFunction(N):
    if N < 0:
        Result = False
    else:
        if N > 100:
            Result = False
        else:
            Result = True
    return Result
```

```
def MyFunction(N):
    if N < 0:
        return False
    else:
        if N > 100:
            return False
        else:
            return True
    return # ignored
```

```
def MyFunction(N):
    if N < 0:
        Result = False
    elif N > 100:
        Result = False
    else:
        Result = True
    return Result
```

```
def MyFunction(N):
    if N < 0:
        return False
    elif N > 100:
        return False
    else:
        return True
    return # ignored
```

```
def MyFunction(N):
    if (N >= 0) and (N <= 100):
        Result = True
    else:
        Result = False
    return Result
```

```
def MyFunction(N):
    if N < 0: return False
    if N > 100: return False
    return True
```

```
def MyFunction(N):
    return (N >= 0) and (N <= 100)
```