CMPSCI 119

Practice Midterm #2 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)

```
X15   = 7
ME262 = 10
F104  = "PLANES AND SPACE"
B17   = ["AIR", 7, "TRAINS", 8.5, 3]
XF85  = ("SPACECRAFT", [4, 1, 3], 5, 6.2)
```

1.	`3.5`	float	`X15 / 2`
2.	`17`	int	`X15 + ME262`
3.	`16.2`	float	`ME262 + XF85[-1]`
4.	`16`	int	`len(F104)`
5.	`3`	int	`len(B17[0])`
6.	`ERROR`		`len(ME262)`
7.	`3`	int	`len(XF85[1])`
8.	`15`	int	`ME262 + XF85[2]`
9.	`ERROR`		`B17 + XF85`
10.	`8.0`	float	`X15 + 1.0`
11.	`[4,1,3,7]`	list	`XF85[1] + [X15]`
12.	`"AIRPLANES AND SPACE"`		`B17[0] + F104`
13.	`[0,1,2,3,4,5,6]`	list	`range(X15)`
14.	`[3,4,5,6]`	list	`range(B17[-1],X15)`
15.	`[1,8,15,22]`	list	`range(1,25,X15)`
16.	`ERROR`		`range(F104)`
17.	`ERROR`		`XF85[0] + ME262`
18.	`9`	int	`len(B17[0]+B17[2])`
19.	`1`	int	`X15 % 2`
20.	`False`	bool	`X15 < 3`
21.	`5`	int	`5 * (X15 > 4)`
22.	`49.0`	float	`X15**2.0`
23.	`[4,1,3,0,1,2]`	list	`XF85[1] + [I for I in range(3)]`
24.	`[-7,0,7]`	list	`range(-X15,X15+1,X15)`
25.	`8.5`	float	`B17[B17[-1]]`
26.	`[0,0,0,0,0,0,0]`	list	`[0 for I in range(X15)]`
27.	`[10,15,9]`	list	`[X15 + I for I in [3,8,2]]`
28.	`["A","I","R"]`	list	`[CH for CH in B17[0]]`
29.	`[1,1,1]`	list	`[len(CH) for CH in B17[0]]`
30.	`["R","I","A"]`	list	`[B17[0][len(B17[0])-1-I] for I in range(len(B17[0]))]`

<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 (F,R,O,G=1):
    global Newt
    Newt = (F * R) + (O * G)
    print (Newt + F)
    return

def FN2 (T,O,A,D):
    global Newt
    FN1 (D+Newt,O,A,T)
    print (Newt)
    return

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

```
# Answer #1    4
#
# Answer #2    18
#
# Answer #3    14
#
# Answer #4    3
#
# Answer #5    7
#
# Answer #6    4
#
# Answer #7    11
#
# Answer #8    6
#
# Answer #9    28
#
# Answer #10   20
#
```

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

```
def FN(W,Q,X=2):
    print (W+Q-X)
    return

def F2(Q,Z,W=3):
    FN(Z,Q)
    print (W+Z)
    return
```

```
def Main():
    A = 4
    B = 7
    FN(A,B)
    FN(2,B,A)
    F2(A,A)
    F2(5,B,A)
    return
```

Answers:

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

<4> 10 Points – Write a complete Python function called **`Guess`**, with no parameters, that use the python **`input`** function to ask the user for a numeric input between 0 and 100 (inclusive). It continues to do so if the user enters anything outside that range, but then returns the entered value when it is inside the correct range.

```
def Guess():
    N = int(input("Enter a number in [0...100]"))
    while (N < 0) or (N > 100):
        N = int(input("Enter a number in [0...100]"))
    return N
```

Scoring: 2 points for `def Guess():`, 2 points for the first `input` statement, 3 points for the `while` loop, 2 points for the second `input` statement, and 1 point for the `return`.

<5> 15 Points – Write a complete Python function called **`Check`** with one integer parameter; the result returned from **`Check`** should be **`"NEGATIVE"`** if the parameter is less than zero, **`"POSITIVE"`** if the parameter is greater than zero, and **`"ZERO"`** if the parameter is exactly zero.

```
def Check(N):
    if (N < 0): return "NEGATIVE"
    elif (N > 0): return "POSITIVE"
    else: return "ZERO"

def Check(N):
    if (N < 0): Answer = "NEGATIVE"
    elif (N > 0): Answer = "POSITIVE"
    else: Answer = "ZERO"
    return Answer
```

Scoring: Accept any solution that accomplishes the desired task; two possible solutions are presented here. Remove 1 point per error (indentation, missing colons or quotes, unbalanced parentheses, use of incorrect operators, incorrect use of parameters, etc.). Note that variable names may be different.

<6> 5 Points – If **`S`** is a string, **`L`** is a list, and **`T`** is a tuple, which of the following expressions are legal, which are illegal, and why?

`S[0] = "X"`	`L[0] = "X"`	`T[0] = "X"`
Illegal, Immutable	Legal	Illegal, Immutable

<7> 10 Points – Write a new code fragment (not a complete program) that creates a list, called **L**, containing exactly 1000 integers, where each integer is computed as the number 100 plus a random value between -50 and +50 (inclusive). You may assume that **import random** has already been written.

```
L = []
for I in range(1000):
    L = L + [100 + random.randrange(-50,51)]
```

-or-

```
L = []
I = 0
while (I < 1000):
    L = L + [100 + random.randrange(-50,51)]
    I = I + 1
```

-or-

```
L = [100 + random.randrange(-50,51) for I in range(1000)]
```

-or-

```
L = [0 for I in range(1000)]
for I in range(1000):
    L[I] = 100 + random.randrange(-50,51)
```

The version with the `for`-loop is the expected answer, but some may use a `while`-loop or a list comprehension (or a combination) instead. Any of these is OK. Remove 5 points if there are major structural errors but something resembles one of the above solutions, remove 1 or 2 points if there are minor syntax errors.

<8> 9 Points – One of these code fragments is not like the others. 5 points: Identify (circle) the fragment with behavior different from the other seven. 4 Points: Explain how it is different.

```
I = 0                          I = 0
while (I <= 9):                while (I < 10):
    print (I)                      print (I)
    I = I + 1                      I = I + 1

for I in range(0,10,1):        for I in range(9,-1,-1):
    print (I)                      print (9-I)

for I in range(10):            for I in [0,1,2,3,4,5,6,7,8,9]:
    print (I)                      print (I)

I = 0                          I = 1
while (I < 9):                 while (I <= 10):
    print (I)                      print (I-1)
    I = I + 1                      I = I + 1
```

Every code fragment prints out the numbers 0 through 9, except for the example in the lower left (5 points), which prints out the numbers 0 through 8 (4 points).

<9> 20 Points – Modify the following code to use **`for`**-loops instead of **`while`**-loops. Use as few Python statements as possible without changing how the function works. (For any code that needs to be modified, circle it and write the replacement code next to it.)

```
def Flood (Canvas, NewColor=white):
    W = getWidth(Canvas)
    H = getHeight(Canvas)
    Y = 0                                          removed and…
    while (Y < H): for Y in range(H):              …replaced (5 pts)
        X = 0                                      removed and…
        while (X < W): for X in range(W):          …replaced (5pts)
            setColor(getPixel(Canvas,X,Y),NewColor)
            X = X + 1                              removed (3 pts)
        Y = Y + 1                                  removed (3 pts)
        repaint (Canvas)              (4 pts for any other syntax errors)
    return
```

To make each `while`-loop into a `for`-loop, the initialization of the variable is removed, the increment is removed, and the `while` is replaced by a `for` that (a) uses the same control variable, and (b) uses a `range` to get the correct span of values.

<10> 20 Points – (10 points each):

A. Rewrite the following code fragment to use an explicit **`for`**-loop.

```
Result = []
Z = 6
while (Z <= 50):
    Result = Result + [Z]
    Z = Z + 3
```

```
Result = []
for Z in range(6,51,3):
    Result = Result + [Z]
```

B. Now rewrite the code fragment as a ***list comprehension***.

```
Result = [Z for Z in range(6,51,3)]
```

Notice that in the **`while`**-loop the presence of the **`<=`** symbol. Normally, this operator is simply a **`<`** symbol, which closely matches how the **`range`** statement works. Since this is a **`<=`**, the corresponding range limit has to be increased from **`50`** to **`51`**.

Scoring: Start with 10 points for each problem where an answer is present (zero points for blank answers). Accept any answer that works and follows the basic guidelines of the questions (that is, determine if the answers generate the correct list, even if they are not quite what I've given here).

Remove points as follows for part A:

- -1 for using **`range(6,50,3)`**
- -1 for **`range(51)`** or **`range(51,3)`** or **`range(6,51)`**
- -1 for omitting the **`Result = []`** statement
- -1 for omitting the **`[]`** around **`[Z]`** (that is, **`Result = Result + Z`**)
- -1 for any and all capitalization errors (**`FOR`**, **`result`**, **`Range`**, etc.)
- -1 per additional syntax error, up to 5

Remove points as follows for part B:

- -1 for using a different range from part A
- -1 for omitting the **`Result =`** portion of the expression
- -1 for any and all capitalization errors (**`FOR`**, **`result`**, `Range`, etc.)
- -1 per additional syntax error, up to 5

<11> 20 Points – What is printed out when **`Main()`** is called?

```
def Apple (M, N):
    P = M + N
    print (P)
    return

def Pear (P, Z, M):
    Apple (P, M)
    Apple (Z, P)
    return

def Main():
    Pear (2, 7, 3)
    Apple ([2, 7], [3])
    Pear ("Dog","Cat","House")
    return
```

Answers	Explanation of Calculation
`5`	`Pear(2,7,3) → Apple(2,3) → print (2+3)`
`9`	`Pear(2,7,3) → Apple(7,2) → print (7+2)`
`[2,7,3]`	`Apple([2,7],[3]) → print ([2,7]+[3])`
`DogHouse`	`Pear("Dog","Cat","House") →` `Apple("Dog","House") → print ("Dog"+"House")`
`CatDog`	`Pear("Dog","Cat","House") →` `Apple("Cat","Dog") → print ("Cat"+"Dog")`

Scoring: 3 points per line (total of 15 points). Remove the 3 points if any line is missing. For the remaining 5 points, assess as follows:

- -1 for any mistakes in calculating **`5`** or **`9`**
- -1 for any mistakes in **`[2, 7, 3]`**
- -1 for any mistakes in **`DogHouse`**, including capitalization
- -1 for any mistakes in **`CatDog`**, including capitalization
- -1 for any extant lines out of order

Notice that the function **`Apple`** doesn't really care about the type of its arguments, as long as addition has some meaning for those types. This is why addition returns a result for integers, lists, and strings.

<12> 20 Points – Write a function called **`Squirt`**, with a string parameter containing a file name and an integer parameter containing a numeric limit, that creates and writes to the file indicated by the file name all integers and their square roots from 0 up through and including the limit.

```
def Squirt(Filename, Limit):
    Outfile = open(Filename, "w")
    for I in range(Limit+1):
        Outfile.write(str(I)+" "+str(math.sqrt(I))+"\n")
    Outfile.close()
    return
```

<13> 5 Points – What is the result in **`L`** of the following code fragment?

```
L = []
for I in range(65,70):
    L + [chr(I)]
```

Answer: **`[]`**
The expression **`L + [chr(I)]`** is evaluated but never assigned to anything, so **`L`** remains unchanged.

<14> 5 Points – What is the result in **`L`** of the following code fragment?

```
L = []
for I in range(65,70):
    L = L + chr(I)
```

Answer: Error, program crashes. You cannot add a list and a string.

<15> 5 Points – What is the result in **`L`** of the following code fragment?

```
L = []
for I in range(65,70):
    L = L + [chr(I)]
```

Answer: **`["A","B","C","D","E"]`**
The expression **`range(65,70)`** gives the list **`[65,66,67,68,69]`**, and since **`65=ord("A")`** and **`"A"=chr(65)`**, the **`chr(…)`** function applied to the range gives the first five capital letters.