CMPSCI 119
Spring 2020
Wednesday, March 4, 2020
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 (students do not have to include the data type):

```
N = 5
X = 2.5
S = "This is a Frog"
L = [3, True, 8.9, [3, 6, 7], "Cat", 2]
T = (4, 9, [2, 8, 3], False, "Dog", 3)
```

Show the computed result for each problem; all are independent of one another. Indicate ERROR in cases where a computation fails. Be careful about the *type* of the result, particularly int, float, bool, and complex types, and put proper quotes around string results, square brackets around lists, and parentheses around tuples.

	Question	Answer	
1.	`N / 2`	`2.5`	`float`
2.	`N / 2.0`	`2.5`	`float`
3.	`N // 2`	`2`	`int`
4.	`X * 2`	`5.0`	`float`
5.	`len(S)`	`14`	`int`
6.	`len(L)`	`6`	`int`
7.	`len(T)`	`6`	`int`
8.	`len(N)`	`ERROR`	
9.	`S[1]`	`"h"`	`string`
10.	`S[-1]`	`"g"`	`string`
11.	`S[:4]`	`"This"`	`string`
12.	`S[10:]`	`"Frog"`	`string`
13.	`L[3][2]`	`7`	`int`
14.	`len(L[-2])`	`3`	`int`
15.	`range(L[0])`	`[0,1,2]`	`list`
16.	`range(len(T))`	`[0,1,2,3,4,5]`	`list`
17.	`range(N,len(S),2)`	`[5,7,9,11,13]`	`list`
18.	`"X"*T[-1]`	`"XXX"`	`string`
19.	`S[0]*N`	`"TTTTT"`	`string`
20.	`[0]*N`	`[0,0,0,0,0]`	`list`
21.	`L+T`	`ERROR`	
22.	`L[3]+T[2]`	`[3,6,7,2,8,3]`	`list`
23.	`L[3][1]+T[2][-1]`	`9`	`int`
24.	`[N+Q for Q in range(N)]`	`[5,6,7,8,9]`	`list`
25.	`[CH for CH in L[-2]]`	`["c","a","t"]`	`string`

<2> 15 Points – What is the final result in **L** of executing the following code fragment?

```
S = "Frog"
L = []
for I in range(len(S)):
    L = L + [S + str(I)]
```

```
["Frog0", "Frog1", "Frog2", "Frog3"]
```

or

```
['Frog0', 'Frog1', 'Frog2', 'Frog3']
```

Scoring:

5 points for a list of strings (remove all 5 points if answer is a single string).
5 points for the correct number of unique items (four).
5 points for the correct configuration of those items (**"Frog"** plus a digit).
Remove 1 point per section for minor errors but do not go below 0 points.

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

```
def Pear(Penny,Dime,Dollar=2):
    Farthing = Dime * Dollar + Penny
    return Farthing + 1

def Apple(Dollar,Quarter=1,Nickel=2):
    return Pear(Nickel,Dollar,Quarter)

def Kiwi(Crown,Nickel,Shilling):
    return Apple(Pear(Crown+Shilling,Nickel))

def Main():
    print (Kiwi(3,0,2))      # Answer #1:  9
    print (Apple(4,3))       # Answer #2:  15
    print (Pear(3,1,4))      # Answer #3:  8
    print (Pear(3,-2))       # Answer #4:  0
    return
```

<4> 20 Points – Complete the following function **`FizzBuzz`** to return the string **`"Fizz"`** if **`N`** is divisible by 3, **`"Buzz"`** if **`N`** is divisible by 5, **`"FizzBuzz"`** if **`N`** is divisible by both 3 and 5, and an empty string if **`N`** is not divisible by either 3 or 5. Your solution must ***NOT*** contain any **`print`** statements. Remember that the **`%`** operator computes the remainder of an integer division.

Scoring: There are a **lot** of correct solutions to this problem, not all of which are listed here. Graders must look carefully at each student's solution to first try to determine what approach is being attempted, and then remove 1 point per error (syntax or logic) for that solution. Remove 5 points for including any `print` statements. Do not go below 0.

```
def FizzBuzz(N):
    if ((N % 3) == 0):
        if ((N % 5) == 0):
            Result = "FizzBuzz"
        else:
            Result = "Fizz"
    else:
        if ((N % 5) == 0):
            Result = "Buzz"
        else:
            Result = ""
    return Result
```

```
def FizzBuzz(N):
    if ((N % 3) == 0):
        if ((N % 5) == 0):
            Result = "FizzBuzz"
        else:
            Result = "Fizz"
    elif ((N % 5) == 0):
        Result = "Buzz"
    else:
        Result = ""
    return Result
```

```
def FizzBuzz(N):
    if ((N % 3) == 0):
        if ((N % 5) == 0):
            return "FizzBuzz"
        else:
            return "Fizz"
    else:
        if ((N % 5) == 0):
            return "Buzz"
    return ""
```

```
def FizzBuzz(N):
    IsFizz = ((N % 3) == 0)
    IsBuzz = ((N % 5) == 0)
    if IsFizz and IsBuzz: Result = "FizzBuzz"
    elif IsFizz: Result = "Fizz"
    elif IsBuzz: Result = "Buzz"
    else: Result = ""
    return Result

def FizzBuzz(N):
    if ((N % 3) == 0) and ((N % 5) == 0): Result = "FizzBuzz"
    elif ((N % 3) == 0): Result = "Fizz"
    elif ((N % 5) == 0): Result = "Buzz"
    else: Result = ""
    return Result

def FizzBuzz(N):
    if ((N % 3) == 0) and ((N % 5) == 0): return "FizzBuzz"
    if ((N % 3) == 0): return "Fizz"
    if ((N % 5) == 0): return "Buzz"
    return ""

def FizzBuzz(N):
    Result = ""
    if ((N % 3) == 0): Result = Result + "Fizz"
    if ((N % 5) == 0): Result = Result + "Buzz"
    return Result
```

<5> 20 Points – Write <u>two</u> loops (one a `while`-loop and the other using a `for`-loop) where in each case the counter variable is called **`Mars`**, the initial value is 9, the test value is 24, and the step value is 5. The payload of each loop is to print ***<u>the square of</u>*** **`Mars`**. Your solutions must **<u>not</u>** contain any **`def`** or **`return`** statements.

`while-loop (10 points)`

```
Mars = 9
while (Mars < 24):
    print (Mars*Mars)
    Mars = Mars + 5
```

`for-loop (10 points)`

```
for Mars in range(9,24,5):
    print (Mars*Mars)
```

5 points – Basic `while`-loop structure must be (-1 point per error):

```
Mars = ____
while Mars < ____:       # parentheses optional
    # payload
    Mars = Mars + ____
```

-1 for **`<=`** instead of **`<`**

Allow **`Mars += _____`** form.

5 points – Basic `for`-loop structure must be (-1 point per error):

```
for Mars in range(____,____,____):
    # payload
```

10 points (5 points per section) – In each of the two sections:

3 points – Blanks must be filled in with **`9`**, **`24`**, and **`5`**, respectively (1 point each).

2 points – Payload must be:

```
print (Mars*Mars)
```

-1 point for **`math.sqrt(Mars)`**

Allow **`Mars**2`** but -1 point for **`Mars^2`** to compute the square.

<6> 5 Points – Short Answer – What are the differences in running time (efficiency) between the two following ways of writing the **SGN** function? Answer on the bottom of this page.

```
def SGN(N):                          def SGN(N):
    if N <  0: Result = -1               if   N < 0: Result = -1
    if N >  0: Result = +1               elif N > 0: Result = +1
    if N == 0: Result = 0                else:       Result = 0
    return Result                        return Result
```

In the left version, **all three `if`-statements are executed** even if the first one is the one that is true; in that case the others need not be executed at all. In the right version, the **first condition that is true terminates** the `if-elif-else` structure immediately.

Therefore, the **left version is less efficient** than the right version and **takes longer to run**.

Scoring:

2 points for any kind of analysis based on how many `if`-statements get executed. Accept anything reasonable.

3 points for stating that the right version is the more efficient approach.