CMPSCI 119
Spring 2019
Final Exam Solution Key
Friday, May 3, 2019
Professor William T. Verts

<1> 15 Points – Answer any **15** questions. Answer more for extra credit. Blank answers will be ignored, correct answers as +1 and incorrect answers as -1. –½ for wrong type in either Result or Type column (for lists, `list` is only thing needed, no need to say `list of type`). For each statement show the computed result and the data type of the result (int, float, bool, list, string, tuple, etc.) Questions are all independent of one another. If a calculation would result in an error, answer **ERROR** in the Result box. Variables have values as follows:

```
Mercury = 59
Venus = 224.7
Earth = "Pale Blue Dot"
Mars = ["Angry", "Red", "Planet"]
JupiterMoons = ["Ganymede", "Europa", "Io", "Callisto"]
Discoveries = {"Uranus":1781, "Neptune":1846, "Pluto":1930}
Moons = [0,0,1,2,79,62,27,14,5]
```

Result	Type	Statement
29.5	float	Mercury / 2
29	int	Mercury // 2
29.0	float	Mercury // 2.0
224	int	int(Venus)
ERROR		len(Venus)
13	int	len(Earth)
3	int	len(Mars)
"Angry"	string	Mars[0]
"P"	string	Earth[0]
"Callisto"	string	JupiterMoons[-1]
"Blue"	string	Earth[5:9]
1930	int	Discoveries["Pluto"]
ERROR		Discoveries[1846]
["Uranus","Neptune", "Pluto"]	list *of string*	Discoveries.keys()
ERROR		Discoveries[Mars[1]]
"Pale Blue Dot Planet"	string	Earth + " " + Mars[-1]
[0,1,2,3,4]	list *of int*	range(Moons[-1])
[59,60,61]	list *of int*	range(Mercury,Moons[5])
[]	list *empty*	range(Moons[4],Mercury)
["I","o"]	list *of string*	[Q for Q in JupiterMoons[2]]
[2, 62, 0]	list *of int*	[Moons[I] for I in [3,5,1]]
[5, 3, 6]	list *of int*	[len(X) for X in Mars]
ERROR		[Z+1 for Q in Earth]
[0,0,0,0]	list *of int*	[0 for Frog in JupiterMoons]
["B","l","u","e"]	list *of string*	[Earth[I] for I in range(5,9)]

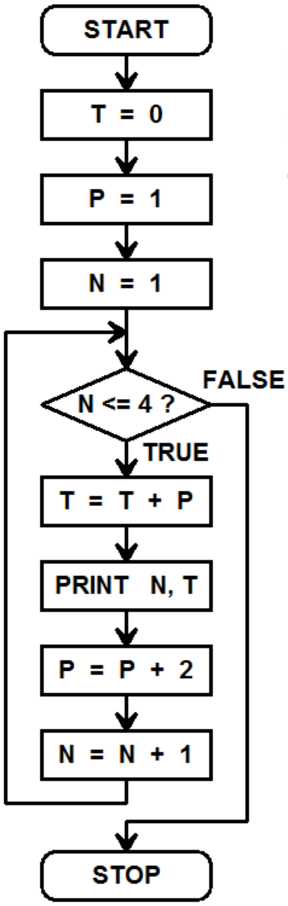
<2> 20 Points – The following flowchart represents a real program.

A: (10 points) "Run" the program below and show how the variables change over time and show what is printed on the output. As each variable is assigned a value, scratch out the old value in the variable box and write in the new value. There will be exactly four lines of output; write the first thing printed next to **LINE #1:**, the second thing printed next to **LINE #2:**, etc..

Scoring: In this section the outputs for the four lines is more important than the variables boxes (although related). Assign 2 points for each of the four output lines, and the remaining 2 points to the variables showing anything close to the correct sequencing.

B: (10 points) Convert the flowchart into an equivalent, runnable Python 3 program. Your result needs only assignment statements, the `print` statement, and the `while`, (no functions or anything fancy). The first statement has been done for you.

Scoring: Assign 1 point for each of the seven lines of code that students have to write; remove ½ point per minor error (forgetting colon or parentheses, indentation, etc.), up to 2 occurrences per line. Assign the remaining 3 points to any overall errors (1 per error) not covered here.



Variables

P ~~1~~ ~~3~~ ~~5~~ 7

N ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

T ~~0~~ ~~1~~ ~~4~~ ~~9~~ 16

```
T = 0
P = 1
N = 1
while N <= 4:
    T = T + P
    print (N,T)
    P = P + 2
    N = N + 1
```

LINE #1: 1,1

LINE #2: 2,4

LINE #3: 3,9

LINE #4: 4,16

<3> 10 Points – Write a new Python function called **Ranger** with three parameters **N1**, **N2**, and **N3**, in that order (and make **N3** have the default parameter value of **1**), that computes and returns a list that starts at **N1**, goes up to but does not include **N2**, in increments of **N3**. Your solution must ***NOT*** use the range function! (That is, you cannot simply use range(N1,N2,N3) – your solution must build the list the hard way!)

```
def Ranger(N1,N2,N3=1):
    Result = []
    Counter = N1
    while (Counter < N2):
        Result = Result + [Counter]
        Counter = Counter + N3
    return Result
```

Scoring: Remove 1 point per minor error, but do not go below zero. Possible errors include, but are not limited to:

Wrong name of function
Wrong parameter list
Omitting default parameter value
Not using a counter loop
Wrong values on counter loop
Omitting brackets on **[Counter]**
Not returning constructed list
etc.

-5 for using the **range** function in any way, even if the result is correct.

<4> 5 Points – Re-write the code below as a list comprehension:

```
L = []
for Z in range(5): L = L + [Z*Z]
```

```
L = [Z*Z for Z in range(5)]
```

Scoring: Remove 1 point per error, but do not go below zero. Possible errors include, but are not limited to:

Using **Z** instead of **Z*Z** as the expression
Forgetting to assign result to **L**
Using a variable different from **L**
Omitting the brackets
Illegal syntax on the **for**-loop part
etc.

<5> 5 Points – In someone's classroom, students get an A for any grade 90 or above, a B for 80 to 90 (that is, grades at least 80 but less than 90), a C for 70 to 80, a D for 60 to 70, and an F otherwise. Complete the code fragment below to print the correct letter grade.

```
Grade = int(input("Enter a grade --- "))
if Grade >= 90: Result = "A"
elif Grade >= 80: Result = "B"
elif Grade >= 70: Result = "C"
elif Grade >= 60: Result = "D"
else: Result = "F"
print (Result)
```

```
Grade = int(input("Enter a grade --- "))
if Grade >= 90: print("A")
elif Grade >= 80: print("B")
elif Grade >= 70: print("C")
elif Grade >= 60: print("D")
else: print("F")
```

```
Grade = int(input("Enter a grade --- "))
if (Grade >= 90): print("A")
if (Grade >= 80) and (Grade < 90): print("B")
if (Grade >= 70) and (Grade < 80): print("C")
if (Grade >= 60) and (Grade < 70): print("D")
if (Grade < 60): print("F")
```

Any of these approaches are OK. Others may be appropriate as well. Accept any approach that gives the correct result. For any legitimate approach, remove 1 point per syntax or logic error, but do not go below zero. Students should receive some credit for any legitimate code that approximates any correct solution.

<6> 5 Points – The following code is full of errors, both syntax and run-time. Find and correct them.

```
~~D~~def Stuff (N):          # N is an integer
    X = N
    while (N > 0):
        print (~~n~~N)
        N = N - 2
    Result = X + N
    if Result == 0:
        Result = Result + 1
→      print ("Error")
→return Result
```

Def should be **def**

Comment is missing leading **#**

Colon missing on end of **while**

print (n) should be **print (N)**

N - 2 should be **N = N - 2**

= should be **==** in **if** statement

Second **print** is at wrong indentation level (increase by one space)

return is at wrong indentation level (increase by four spaces)

Scoring: Remove ½ point for every mistake not found (total of 4 points), and remove ½ point for every correct item misidentified as a mistake, but do not go below zero.

<7> 15 Points – What is printed by the following code when **`Main`** is run?

Scoring: 3 Points per answer.

```
def F1(Frog):
    global X
    Result = Frog + X
    X = X + 1
    return Result

def F2(M,N=3):
    global X
    X = X - 2
    Result = F1(M + N + X)
    return Result

def Main():
    global X
    X = 1
    F0 = lambda Q : Q+2
    print (F1(3))          # Result 1:     4
    print (F2(3))          # Result 2:     6
    print (F2(1,1))        # Result 3:     0
    print (F2(1,1))        # Result 4:    -2
    print (F0(5))          # Result 5:     7
    return
```

<8> 10 Points – The two code fragments below are almost but not quite the same. What is printed out by each one?

```
try:
    print (10//2)
    print (10//1)
    print (10//0)
    print (10//2)
except:
    print ("Error")
```

```
5
10
Error
```

```
try:
    print (10//2)
    print (10//1)
    print (10//1)
    print (10//2)
except:
    print ("Error")
```

```
5
10
10
5
```

Scoring: 5 points for each set of answers.
Remove 2 points for printing `5-10-10-5-Error` in either set of answers.

<9> 5 Points – Fill in the blanks to write all numbers from 1 through 1000, one per line, to the file named in **`Filename`**:

```
def Creator(Filename):

    Handle = open(Filename, "w")

    for I in range(1,1001):

        Handle.write(str(I) + "\n")

    Handle.close()

    return
```

Scoring: 1 point for each line. Remove ½ point per error (max two of these per line), including but not limited to:

- Not using **`Filename`** in first parameter of **`open`**,
- Not specifying **`"w"`** in second parameter of **`open`** (or forgetting quotes),
- Wrong starting value in **`range`**,
- Wrong ending value in **`range`**,
- Not using **`str()`** in **`write`**,
- Forgetting the **`"\n"`** in **`write`**,
- Using **`,`** instead of **`+`** in **`write`**,
- Forgetting **`()`** in **`close`**,
- etc.

<10> 10 Points – Short Answer – Please use the back of this page for your answer.

A. (5 points) Can I always write a **`for`**-loop as a **`while`**-loop? Why or why not?

YES (2 points)

A **`for`**-loop always uses a list to control the number of times through the loop and what values are used in each pass; lists have a finite number of elements. For example:

```
for MyVar in MyList:
    # Do loop payload with MyVar
```

This can always be written with a counter loop in a **`while`**-loop that steps through the same list sequentially, and from the counter we can extract the appropriate item from the list:

```
Counter = 0
while (Counter < len(MyList)):
    MyVar = MyList[Counter]
    # Do loop payload with MyVar
    Counter = Counter + 1
```

(3 points for a reasonable explanation, 2 points for an almost correct explanation, 1 point for anything marginally appropriate.)

B. (5 points) Can I always write a **`while`**-loop as a **`for`**-loop? Why or why not?

NO (2 points)

A **`while`**-loop can run an arbitrary and unknown number of times based on whether the loop condition is controlled by user input, some random process, or is infinite. A **`for`**-loop can't do any of that. For example:

```
while True:
    # Do loop payload

while int(input("Enter a positive number -- ")) > 0:
    # Do loop payload

while random.randrange(100) != 0:
    # Do loop payload
```

(3 points for a reasonable explanation, 2 points for an almost correct explanation, 1 point for anything marginally appropriate.)