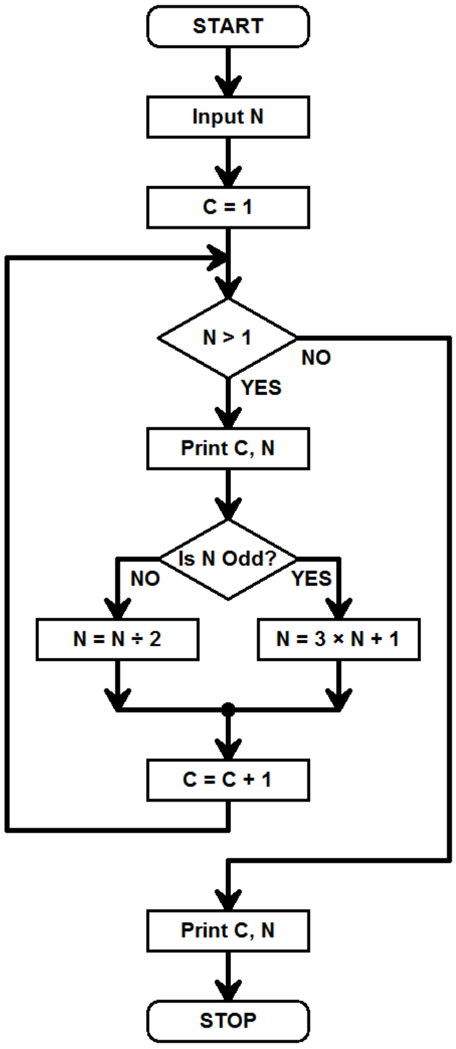
CMPSCI 119
Fall 2018
Final Exam Solution Key
Thursday, December 20, 2018
Professor William T. Verts

<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. In each question there is an expression and the desired result computed when that expression is evaluated, but every statement contains an unknown value with unknown data type. Fill in the blanks with the value that causes the desired result (and data type) to be computed. Questions are all independent of one another. Variables have values as follows:

```
Donner = 23
Blitzen = 7.6
Comet = "Eight Tiny Reindeer"
Cupid = 15L
Rudolph = makeColor(255,0,0)
Sleigh = {5:"Elf", 9:"Toy", 2:"Workshop"}
Chimney = [6, "Bag", 9, 2.3, 5, "North Pole"]
```

Statement	Desired Result
Donner / 2	11
Donner / 2.0	11.5
Donner + 7L	30L
Cupid + 5.0	20.0
(Donner + Cupid) * 1.0	38.0
Comet[6: 10]	"Tiny"
Comet[: 5]	"Eight"
Comet[11 :]	"Reindeer"
"S" + Comet[1 :5]	"Sight"
Blitzen + Donner / 3	14.6
int(Blitzen) + Donner / 3	14
int(Blitzen) + Donner / 2.0	18.5
getRed(Rudolph) * 2.0 *or* Error	510.0
len(Sleigh[2])	8
Sleigh[6+ 3]	"Toy"
Sleigh[Donner/ 4]	"Elf"
len(Chimney[1])	3
range(1,Donner, 5)	[1,6,11,16,21]
range(3,len(Comet)+ 5 ,4) *or* 6 *or* 7 *or* 8	[3,7,11,15,19,23]
range(int(round(Blitzen/ 2)))	[0,1,2,3]
[Comet[I] for I in range(5)]	["E","i","g","h","t"
[0 for I in range(Donner - 17)]	[0,0,0,0,0,0]
Donner + Chimney[4]	28
Chimney[4] * 2.0	10.0
Chimney[28 - Donner] *or* 22	"North Pole"

<2> 10 Points – The following flowchart needs to be converted into an equivalent, runnable Python program. Complete the code. Your result needs only assignment statements, the **`print`** statement, **`while`**, and **`if`** (no functions or anything fancy). The first two statements have been done for you.



```
N = input("Enter N --- ")
C = 1
while (N > 1):
    print C,N
    if (N % 2) == 1:
        N = 3 * N + 1
    else:
        N = N / 2
    C = C + 1
print C,N
```

-½ point per minor syntax error
-1 per major error
Do not go below zero.

<3> 5 Points – Write a new Python function called **`Whatsit`** with three parameters **`Z1`**, **`Z2`**, and **`Which`**, in that order (and make **`Which`** have the default parameter value of **`True`**), that computes and returns the value **`Z2 - Z1`** only if **`Which`** is **`True`**, and returns the value **`Z1 - Z2`** otherwise.

Any of these solutions are OK:

```
def Whatsit (Z1,Z2,Which=True):
    if Which:                                    or if Which == True:
        return Z2 - Z1
    else:
        return Z1 - Z2

def Whatsit (Z1,Z2,Which=True):
    if Which:                                    or if Which == True:
        Result = Z2 - Z1
    else:
        Result = Z1 - Z2
    return Result

def Whatsit (Z1,Z2,Which=True):
    if Which: return Z2 - Z1                     or if Which == True:
    else:     return Z1 - Z2

def Whatsit (Z1,Z2,Which=True):
    if Which: Result = Z2 - Z1                   or if Which == True:
    else:     Result = Z1 - Z2
    return Result

def Whatsit (Z1,Z2,Which=True):
    Result = Z1 - Z2
    if Which: Result = -Result                   or if Which == True:
    return Result
```

1 point for **`def Whatsit`**
1 point for parameter list
1 point for **`if-else`**
1 point for **`return`** *`value`*
1 point for correct values

<4> 5 Points – What is the final result printed by the code below?

```
L = ["Frog","Toad","Cat","Aardvark","Snake","Dog"]
T = 0
for Z in L: T = T + len(Z)
print T
```

27 (sum of the lengths of all strings in the list)

<5> 5 Points – What are the results of each of the following two code fragments?

```
X = 0
if (X < 10):
    X = X + 1
print X
```

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

1 (2 points) **11** (3 points)

<6> 5 Points – Here are two versions of the same function. Which one is the recursive version? What is the result of `Weird(4)` (either version)?

```
def Weird (N):
    if N == 0: Result = 1
    else: Result = Weird(N-1) * 2
    return Result
```

← **Recursive** (2 points)

```
def Weird (N):
    Result = 1
    for I in range(N): Result = Result * 2
    return Result
```

16 (3 points)

<7> 15 Points – 5 points each – What is printed by the following code when **Main** is run?

```
def F1(X):
    return X+4

def F2(M,N=3):

    def F3 (A):
        return F1(A)+1

    return F3(M+N)

def Main():
    print F1(3)          # Result 1: ____7____
    print F2(3)          # Result 2: ____11___
    print F2(1,1)        # Result 3: ____7____
    return
```

<8> 15 Points – Write a simple function to increase the contrast of an image. For each pixel in **Canvas**, the new red value is the integer part of the difference between the pixel's old red value and the average red passed in, times 1.5, plus the average red. The new green and new blue values are computed similarly. You will need **getRed**, **setRed**, **getGreen**, etc., but no other JES functions.

5 points per color primary. Remove 1 point for each syntax and logic error, but do not go below zero. It is OK if statements are spread across several lines.

```
def Contrast(Canvas,AverageR=128,AverageG=128,AverageB=128):
    for Y in range(getHeight(Canvas)):
        for X in range(getWidth(Canvas)):
            PX = getPixel(Canvas,X,Y)

            setRed  (PX,int((getRed  (PX)-AverageR)*1.5+AverageR))
            setGreen(PX,int((getGreen(PX)-AverageG)*1.5+AverageG))
            setBlue (PX,int((getBlue (PX)-AverageB)*1.5+AverageB))

        repaint(Canvas)
    return
```

<9> 5 Points – Short Answer – We have explored two different approaches for representing polynomials in Python. One represents each term as a list of coefficient and exponent, and has a list of those lists. The other uses a single list where the exponent is the index into the list of coefficients. For example, the polynomial $4x^2 - 3x + 6$ would be represented as `[[4,2], [-3,1], [6,0]]` in the first case, and as `[6, -3, 4]` in the second case. What are the benefits and drawbacks of each approach? If I chose the second approach as my data structure, give an example of a polynomial that would not work as well as in the first approach.

3 points – In the coefficient-and-exponent approach, sparse polynomials would be efficiently represented but the code to handle them would be complicated, whereas the exponent-as-index approach is very efficient to represent small, packed polynomials and the code to handle them is simple, but sparse polynomials with large exponents would not be very efficient (Accept anything reasonable here.)

2 points – An example is $5x^{1000000} + 3x^{500000} - 7$

<10> 5 Points – Finish the **GetANumber** function below to display the **Message** string to the user, ask for a number (either int or float), check the number to see if it is at least **Low** but no bigger than **High**. If not, it will ask again, and continue to ask as many times as the value entered was either less than **Low** or greater than **High**. When the user enters a number in the range then return it as the variable **Result**. (You will use this function in the next problem.)

```
def GetANumber(Message,Low,High):
    Result = input(Message)
    while (Result < Low) or (Result > High):
        Result = input(Message)
    return Result
```

-or-

```
def GetANumber(Message,Low,High):
    Result = Low - 1
    while (Result < Low) or (Result > High):
        Result = input(Message)
    return Result
```

-1 per syntax error.
-1 for using literal string in **input** statements
OK to use JES **requestNumber** function instead of **input**

<11> 10 Points – In a short YouTube clip from "The Avengers" movie, bad-guy Loki throws good-guy Tony Stark out of a window on Stark Tower. As he falls, his fancy new Iron Man suit flies out of the window, catches up to him, envelopes him, and allows him to survive by firing the repulsors at the last possible moment. Stark went out of the window and started falling at time 3:07 in the clip, and fired the repulsors to stop falling at 3:30.

Finish the Python program to compute how tall Stark Tower must be. You will need the physics formula $d = \frac{1}{2}at^2$, where d is distance, a is the acceleration due to gravity (approximately 9.8 m/sec^2 or 32 ft/sec^2), and t is the elapsed time (in seconds). The **Main** program must first ask for the starting and ending number of seconds by calling the **GetANumber** function from the previous page with an appropriate **Message** string, a **Low** value of 0, and a **High** value of 1000. It must then compute the elapsed time in seconds, compute the distance in both meters and in feet, and print out the elapsed time and the two distances.

```
def Main():
    Start = GetANumber("Enter Start Time --- ", 0, 1000)
    Stop  = GetANumber("Enter Stop Time --- ", 0, 1000)
    Elapsed = Stop - Start
    Feet = 32 * Elapsed * Elapsed / 2.0
    Meters =  9.8 * Elapsed * Elapsed / 2.0
    print Elapsed
    print Feet
    print Meters
    return
```

-1 per syntax error.
-1 for each case of not following directions, such as 0…1000 in **GetANumber**, etc.

<12> 5 Points – How tall is Stark Tower? Is that number reasonable, or is it a Hollywood exaggeration?

3 points – 8464.0 feet, or 2592.1 meters. Accept either.

2 points – Exaggeration! The building is a mile and a half tall!