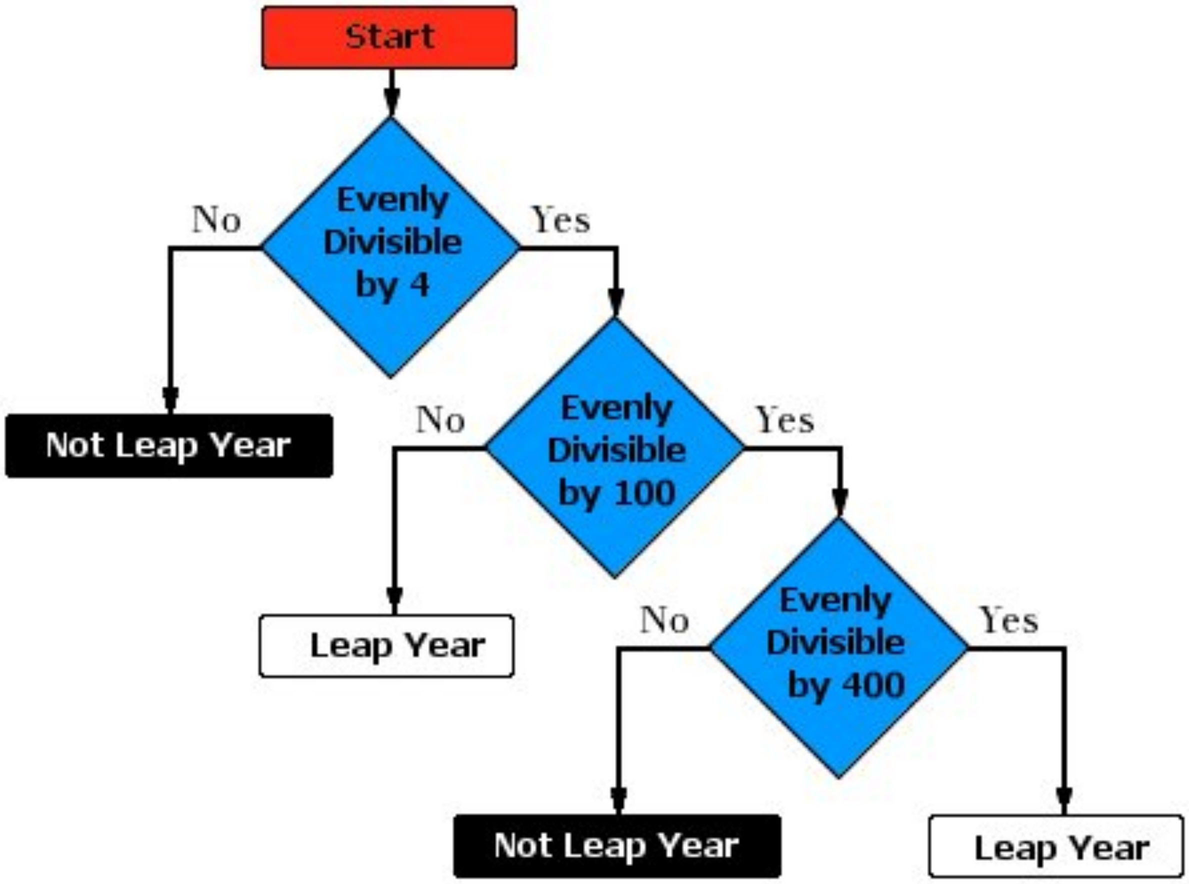
ENG-101 Section I
Intro Computing Engineers

Due: 29 August 2018 at 6:00PM.

Question 1 (20 Points)

Write a well-documented Python program that computes the number of leap years occurring between a range of years specified by a user of the Python program. In general, a leap year happens if the year is divisible by four – except for centuries that are not divisible by 400. Years 1600, 1604, 1608 are leap years, while 1700, 1800 and 1900 are not. The year 2000 is a leap year.



Your solution should be a Python text file *leapYear.py*. Use the IDLE editor to create the file. Your first comment should be your name. In the last line of the Python text file, please provide an answer to the question, “How many leap years occur between the years 1600 and 3200?”, as a comment.

<ins>Hints</ins>: At first, design your code to test for one year, an integer. The logical test can be implemented with the use of three if expression(s), each using Python modulus operator for the relational tests (divisible by 4, 100 and 400). Once your code runs correctly for a single year, extend your result by having your Python software accept a range of years, that is two separate years, both again are integers. Use a for-loop to embed the code that was designed for testing a single leap year. Count the number of leap years that are found in the range. Don’t forget to <ins>initialize</ins> the counter before the for-loop.

Question 2 (20 Points)

Write a well-documented Python program that adds the first N integers together. That is, if N =5, your application should calculate 1+2+3+4+5 =15. Check to see if your software produces the correct result for a few test cases, say N=10, N=20, and N=30. Your answers should be 55, 210 and 465 respectively. Use a for-loop with the specification for j in range (N+1) to include the N^{th} integer in the summation. Before the loop is entered, an accumulator should be initialized to zero. Inside the loop, the accumulator variable should be incremented with each iteration with an expression like accumulator +=j.

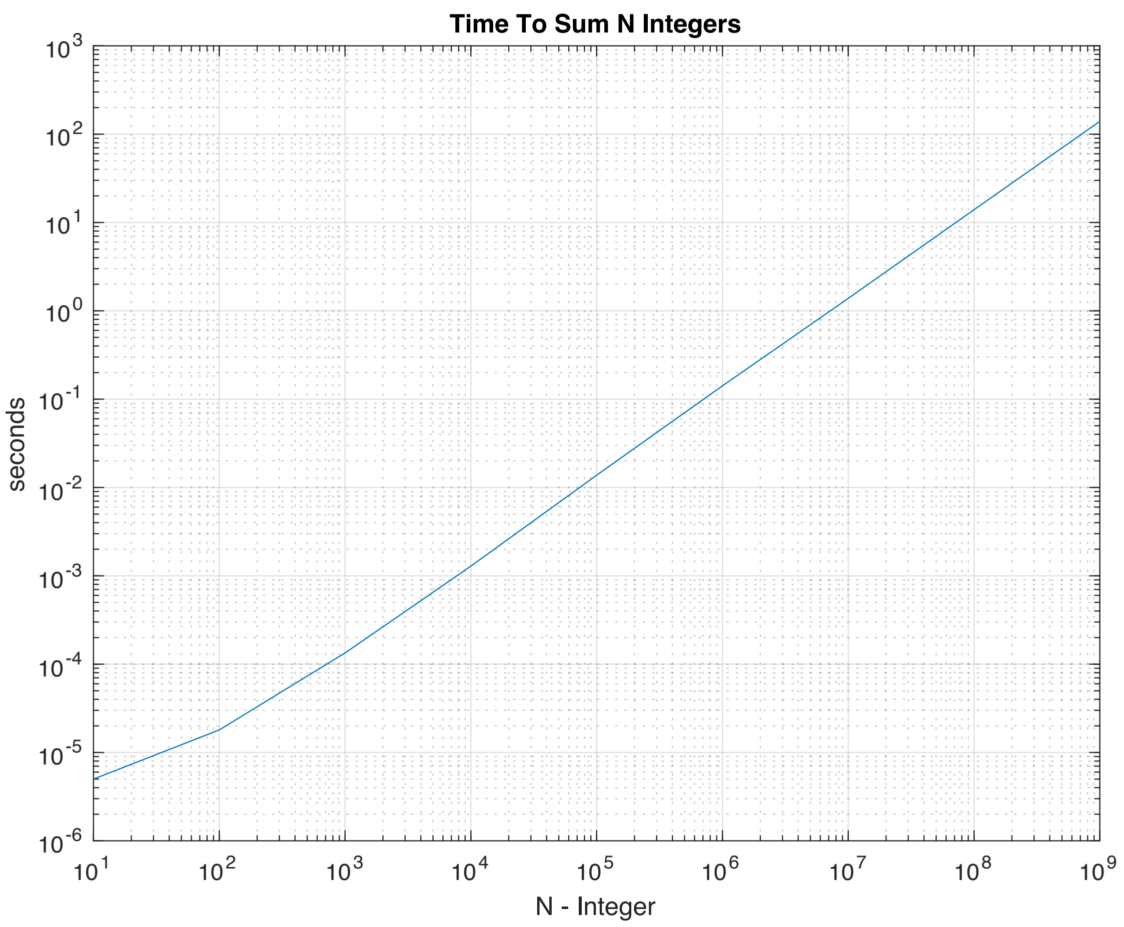
Your solution should be a Python text file *sum.py*. Use the IDLE editor to create the file. Your first comment should be your name.

After you program *sum.py* correctly calculates the sum, extend the program by prompting the user for an integer M. Your program should calculate the integer sums from 1 to 10, 1 to 10**2 (100), 1 to 10**3 (1000), … 1 to 10**M. Each integer step in M increases the summation by a factor of ten. Once your code is running, we do not need to see the result of the summation; rather we are more interested in the time it takes for the code to calculate the sums. As M>9, the summation takes a long time (~ 100 seconds) so be careful in the value of M that your program is run. Your solution should be a text file *sumM.py*. Your first comment should be your name.

Hints: Embed the code that worked to tally the sum of a single integer N – within a for-loop that ranges over i in range(1,M). Adjust the inner-loop over j to read for j in range (10**i+1). Time each outer loop iteration (over i) to tally the time it takes to sum from 1 to 10, then 1 to 100, and so forth. Use the time module available in Python.

The syntax for the code is *from time import time*. Before each outer loop iteration, set a tic = time(). After the summation is complete set a toc = time(). The function time() is the current time of your computer. The execution time of each summation is simply toc-tic.

Graph, using Microsoft Excel or its equivalent, the time it takes to run the sums 1-10, 1-100, 1-100, … 1-1,000,000,000, by having your program accept integer M=9. Given the wide range of sums and processing times, I used a logarithmic plot – see below. Your graph will be different depending on the processing speed and memory available on your computer. Your graph should be submitted in a pdf file *sumgraph.pdf*.



Eric's sumgraph.pdf (file to hand in)

Next Steps (not graded, not to be handed in): In the Python, virtual machine there is no limit to the size of the integers that can be processed. The Python virtual machine extends the length of the hardware registers and pieces together its computation by segmenting the integer into pieces that can be manipulated in hardware.

My result (see above) indicates that the processing time scales with the number of integers being summed. Summing 100 millionth integer (10^8) takes ~10 (10^1) seconds, while summing a billion integers (10^9) takes ~100 (10^2) seconds. Extrapolating our result suggests that summing 10 billion (10^{10}) integers will take 1000 (10^3) seconds, 100 billion (10^{11}) integers will take 10,000 (10^4) seconds, and so forth.

Four months, a third of a year, is approximately (10^7) seconds. Our results suggest that we could sum integer 100,000,000,000,000 (10^{14}) digits within this time. How is it possible for me to know that this sum is 5.00000000000005e+27 without waiting a third of a year?