Lab #6 – Accumulator Loops

*Notes: Copy the* ***Lab6*** *folder from the L drive and save all your work in this folder.*

Part I: Understanding the Accumulator Design

1. There is a program called futvaldesign.py in the Lab6 folder. It has the code from futval.py that Zelle introduced in chapter 2. Fill in the answers to the design questions in the program. You don’t have to change any code.
2. Use the penniesdesign.py program to help you design and code the pennies problem from the Lab #4. Answer the design questions first, then write the code. How similar is the code to what you wrote in Lab #4?

Part II: Some Simple Accumulator Loops

*Note: Use the Accumulator Loop design for each of these problems. You don’t have to explicitly put the answers to design questions in the code; but if you get stuck, it might help.*

1. There is a theorem in math that the sum of the first n numbers is:

Write a program (**sumcheck.py**) that asks the user for some value for n and then prints the result of the sum (1+2+…+n) and the equation on the right. Of course these should be the same. Here’s an example of what running you program should look like:

Check sum of all integers from 1 to n

What value of n? 200

The actual sum is 20100

The formula gives 20100.0

1. Write a program (**oddsum.py**) that sums all the odd numbers up to (and including) a given number. Here’s an example of what it should look like when you run your program.

Sum all the odd values

Enter the starting value (must be odd): 1

Enter the ending value (should be odd): 15

The total is 64

Add a counter to your program. A counter is just an accumulator variable that is initialized to 0 and is incremented (has 1 added to it) each time through the loop. When you are done with the loop, print both the sum of the values and the counter value. So your program should look like this:

Sum all the odd values

Enter the starting value (must be odd): 29

Enter the ending value (should be odd): 99

This program summed 36 values

and the total is 2304

1. Create a program that (**suminverses.py**) sums the values from 1/1 to 1/n. The user will enter the value of n just like the programs above.

Mathematically this is:

Your program should look like this when completed:

Sum the values of 1/i for i from 1 to n (inclusive)

Enter the value for n: 10

The sum is 2.9289682539682538

1. Make a program that calculates the following sum:

Since we can’t actually sum all the values to infinity, let the user input a value and print the result of the summation. Write a program (**sum1overpower2.py**) that implements the above sum. So here’s an example of running the program:

Sum the values of 1 + 1/2 + ... + 1/2\*\*n

Enter the value for n: 10

The sum is 1.9990234375

What interesting characteristic do you notice as you run the program with very large values?

Part II: Implementing Taylor Series for Calculating Math Functions

**Introduction**: Python provides a math library that gives functions that will calculate many of the common trigonometric functions. One way that these values can be calculated is to use a mathematical technique called the Taylor series. The Taylor series involves summing a given mathematical formula an infinite number of times. This, of course, is impossible. However, in the cases that we are going to look at, the terms get smaller and smaller and can eventually be ignored.

**Design and Implementation**

1. For each of the Taylor series given below, design a loop. Use **n** for the number of times to run the loop, since you clearly can’t go to infinity. This means that **i** will take the values from **0** to **n-1**. The **expTaylor.py** program is already completed for you.

**expTaylor.py:**

**sinTaylor.py:**

**is (-1)\*\*i / math.factorial( 2\*i+1 ) \* x\*\*( 2\*i+1 )**

**cosTaylor.py:**

**is**

Formulas above are from [*http://en.wikipedia.org/wiki/Taylor\_series*](http://en.wikipedia.org/wiki/Taylor_series)*:*

1. Implement the designs you have created. Use the procedure that we talked about in class for converting the formulas above to python statements.
2. Add two prompts before the loop to get the values for **x** and **terms**.
3. Run the programs and fill in the rest of the tables below for the given values of **x** and the number of times to run the loop (**terms**). You only have to write the number to 4 decimal places:

**expTaylor.py:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | n=3 | n=5 | n=10 | Actual Value |
| x = 0.0 | 1.0 |  |  | 1.0 |
| x = 0.5 | 1.6458 |  |  | 1.6487 |
| x=1.0 | 2.6667 |  |  | 2.7183 |
| x=1.5 | 4.1875 |  |  | 4.4817 |

*Note that the n values are different below – the sine and cosine series converge much faster than the exponent above. One value is given for each so you can check your results. If you get a different value, go back and check the expression that you have used.*

**sinTaylor.py:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | n=2 | n=3 | n=5 | Actual Value |
| x = 0.0 |  |  |  | 0.0 |
| x = 0.5 |  |  |  | 0.4794 |
| x=1.0 | 0.8417 |  |  | 0.8415 |
| x=1.5 |  |  |  | 0.9975 |

**cosTaylor.py:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | n=2 | n=3 | n=5 | Actual Value |
| x = 0.0 |  |  |  | 1.0 |
| x = 0.5 |  |  |  | 0.8776 |
| x=1.0 | 0.5417 |  |  | 0.5403 |
| x=1.5 |  |  |  | 0.0707 |

1. Put your name on this page and hand in only this page. Submit all the code on CMS.