**COSC 1336, Lab 4 Instructions, Repetition Structures**

**Part 1a:** This part draws boxes. Prompt for and input a value: box\_size, then draw a box of that size. Allow the user to keep drawing boxes until the box\_size is 0. Test examples:

box\_size=3

\*\*\*  
\*\*\*  
\*\*\*box\_size=5

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
box\_size=4

\*\*\*\*  
\*\*\*\*  
\*\*\*\*  
\*\*\*\*

To save space, 3 boxes are shown across. You do not have to draw multiple boxes on one row as shown above. Instead, draw each new box vertically below the previous box.

**Part 1b:** This part enhances Part 1a. Copy and modify the code from part 1a so that even values of box\_size draws solid boxes (as before), but odd values of box\_size draws box outlines. As before, allow the user to draw boxes until the box\_size is 0. Test output:

box\_size=3(outline)

\*\*\*  
\* \*  
\*\*\*box\_size=5(outline)

\*\*\*\*\*  
\* \*  
\* \*  
\* \*  
\*\*\*\*\*  
box\_size=4(solid)

\*\*\*\*  
\*\*\*\*  
\*\*\*\*  
\*\*\*\*

**Part 2a:** This part draws triangles. Prompt for and input a value: triangle\_size, then draw a triangle of that size. Allow the user to keep drawing triangles until 0 is entered. Examples:

triangle\_size=3

\*  
\*\*  
\*\*\*triangle\_size=5

\*  
\*\*  
\*\*\*  
\*\*\*\*  
\*\*\*\*\*  
triangle\_size=4

\*  
\*\*  
\*\*\*  
\*\*\*\*

**Part 2b:** This enhances part Part 2a. Copy and modify code from part 2a so triangles are drawn from size 1 to the largest triangle size. Ask for one number. Draw triangles from size 1 up to and including the requested triangle size. For example, if the user enters 3, draw 3 triangles of increasing size:

\*  
\*  
\*\*  
\*  
\*\*  
\*\*\*

**Part 3a:** This part computes statistics on Austin temperatures. Prompt for and get temperatures from the user. The temperatures can be positive, zero, or negative. Keep a count of how many temperatures are entered.

To indicate that the list is done, allow the user to type in Q or q for quit. To do this, first check if the entry is ‘Q’ or ‘q’; quit if so. If the entry is NOT Q or q, then assume the entry is an integer. You can use temp.lower() == ‘q’ rather than temp==’q’ or temp==’Q’ as a check.

At the end of the list (when the user enters Q or q) print out the following statistics:

* number of temperatures entered
* highest temperature
* lowest temperature
* average temperature
* number of freezing temperatures (less than or equal to 32)

**Important: This should work if zero, one, or 100s of temperature are entered. Perform your final test on the following six temperatures: 12, 23, 45, 56, 78, 89, q**

**Part 3b:** This is an enhancement to the work you did in Part 3a. The program in Part 3a allows whacko temperatures like: -950 or 478 to be entered. Add a validation loop to prevent invalid temperatures from being entered. Ask the user to enter temperatures in the range of -5 to +115. (Recorded Austin temperatures are from -2 to 110F.). If the temperature is NOT in that range, print an error message and keep asking the user to keep entering a temperature until it is valid (or Q or q for quit). See: **Input Validation Loops** in the textbook on pages 147– 152 (3ed); 185 – 190 (4ed).

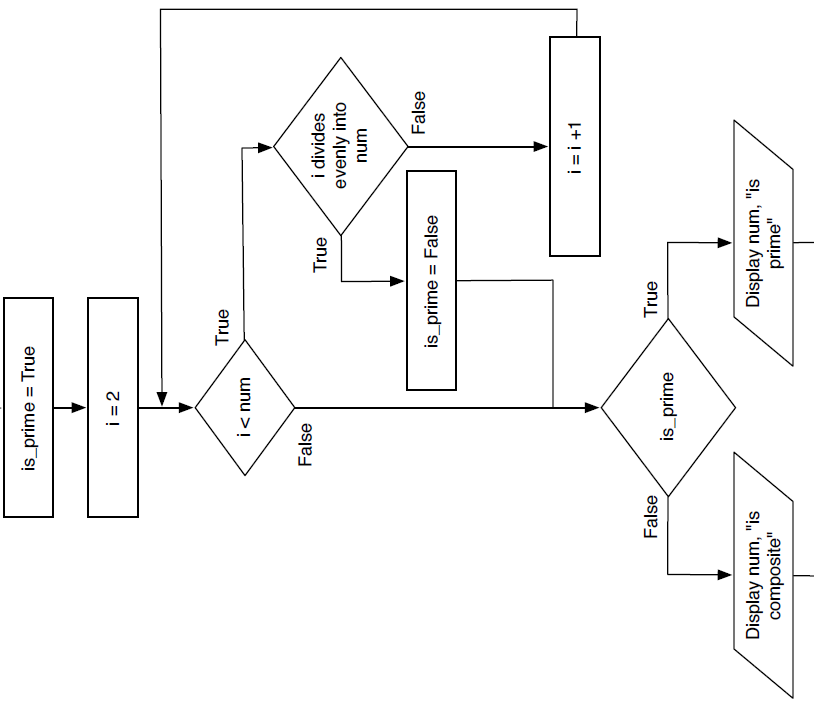
**Keep count of the number of invalid temperatures entered.** Do NOT include invalid temps in the statistics! **Add the following items to the statistics** that you print at the end:

* number of valid temperatures entered *(add this for part 3b)*
* number of invalid temperatures entered *(add this for part 3b)*

**Perform your final test on the following inputs: 12, 45, 78, 123, 456, -4, 0, 99, Q**

**Part 4:** Rigorous use of repetition structures can make difficult tasks easy. Consider how difficult it is for a person to determine whether a number is prime. A prime number can only be divided by 1 or itself. Here are the first few primes: 2, 3, 5, 7, 11, 13, 17, 19, 23. These are easy to recognize as prime. But is the number 9,008,711 prime? Not so easy to tell.

You can repeatedly divide a number, starting at 2, with ever larger numbers, up to one less than the number itself. If NONE of these divide the number evenly, you have a prime. The flowchart looks like this:



For Part 4, write code that asks for a positive integer (num) and implements the flowchart above. Loop the program, so it tests if numbers are prime until you enter 0 to quit, like this:

**Enter a positive integer to see if it is prime (0 to quit): 25  
25 is composite  
Enter a positive integer to see if it is prime (0 to quit): 17  
17 is prime  
Enter a positive integer to see if it is prime (0 to quit): 0  
Goodbye.**  
  
Test this code on 9,008,711. Is it prime?

**Part 5.** This part is easy. Pick any one of the turtle graphics designs from section 4.8: Turtle Graphics: Using Loops to Draw Designs; pages 197 – 300 (4ed). The code is available on Blackboard under Publisher Source 4ed. If time and interest permits, you can modify color and design, but you don’t have to. You can save this as file: **Part5\_graphic.py**

**Part 6.** After you are certain each part works, combine all 6 parts above into one program. Call the file: **DHH\_L4\_Lastname\_parts1-6.py. Submit this one file on Blackboard.**

**While working on each part, you can save it separately:**

**Part1\_boxes.py**

**Part2\_triangles.py**

**Part3\_temps.py**

**Part4\_prime.py**

**Part5\_design.py**

**In Part 6, you combine these 5 parts into one program to submit for grading:**

**DHH\_L4\_Lastname\_parts1-6.py**

**Extra Credit 1:** At the top of the combined program, allow the user to select the option:  
1) boxes, 2) triangles, 3) temps, 4) prime 5) design or 6) quit. After one of the options is selected, perform that task, and then loop back up to the main option prompt. Loop until the user enters 6 to quit. This is a nice enhancement. It makes the program easier to test. After getting the user’s option, provide one elif statement for each option. If an invalid option is entered, display an error message and let the user try again. Use one else statement at the bottom to “catch” an invalid option. This makes the whole program much easier to use.

**Extra Credit: 2** Finding prime numbers using the provided flow-chart is slow. You can make it run faster. For example, even number are never prime. You can increment the test\_number by 2 instead of 1. (test\_number = 3, 5, 7 …) Other optimizations apply. See if you can make it run faster. **Document your optimizations in comments in the prime number detector code.**

**Extra Credit: 3:** Part 4 determined if a number is prime (True or False). Using this work, determine the nth prime number. Ask the user to enter a number n and find the nth prime. Consider this table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| nth prime | 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 |

Ask for the number n, then calculate and output the nth prime. Example: if the user enters the number 10, compute the 10th prime (which is 29) and output 29. This requires another loop.

Tips for extra credit 3: Use the code from Part 4 to detect if a number is prime. After the user enters n (the nth prime number to find), initialize test\_number= 2 and prime\_count=0. , see if test\_number (2) is prime; it is, so increment prime\_count to 1. Then increment test\_number to 3 and see if it is prime; it is, so increment prime\_count to 2. Then increment test\_number to 4, which is not prime, so leave prime\_count at 2 and only increment test\_number to 5. Since 5 is prime, increment prime\_count to 3. Keep doing this, until prime\_count is equal to n. When prime\_count has reached n, you have found the nth prime number, which is test\_number.

This requires nested loops. The program asks for and prints the nth prime until the user enters 0 to quit. Test on 1000. That is, what is the 1000th prime number? It is interesting to see how long it takes Python to calculate the 1000th prime number.

If you did extra credit 1 and extra credit 3, add another option to the main menu. Your main menu prompt will be:

Enter option: 1) boxes, 2) triangles, 3) temps, 4) prime 5) design 6) nth\_prime 7) quit

This lab is a lot of work. Break it down. Do one step at a time. Start with boxes, then move on to other steps. Save intermediate code as you move to the next part. Do not wait until the due date to start this. This lab involves a lot of programming with input, output, decisions and loops.