CSC 361E: Generic Programming in Python, Lab Exercise 1

Question 1

The Problem

We are going to do some conversions, from integer to binary and then from binary back to integer. It will give us a chance to play with *if-elif-else* and *while* statements, as well as a little string slicing. Read the slides on control structures.

Your Task

You prompt for an integer, convert the integer to a binary number string (there is no type for actual binary numbers so we just represent it as a string). We then take the string and turn it back into a regular integer. Things to remember

1. If the integer is 0, then we are done since conversion back and forth of 0 is still 0. The program simply prints a note saying it is 0 and quits.
2. If the integer is negative, then we probably don't know how to do it, so the program prints a message saying it is negative and quits.
3. Otherwise, we do the conversion of the integer to a binary string (a string of 1's and 0's) and then convert that same string back to an integer to make sure we did it right.

Hints

How do we get a binary string from an integer? The easiest method uses integer division by 2 on successive quotients and then collects the remainders. It is best illustrated by an example. Consider the decimal number 156.

- 156 divided by 2 gives the quotient 78 and remainder 0.
- 78 divided by 2 gives the quotient 39 and remainder 0.
- 39 divided by 2 gives the quotient 19 and remainder 1.
- 19 divided by 2 gives the quotient 9 and remainder 1.
- 9 divided by 2 gives the quotient 4 and remainder 1.
- 4 divided by 2 gives the quotient 2 and remainder 0.
- 2 divided by 2 gives the quotient 1 and remainder 0.
- 1 divided by 2 gives the quotient 0 and remainder 1.

Stop at reaching a quotient of 0. The binary equivalent is given by concatenating the remainders, in reverse (so the last remainder is the most significant bit and the first is the least). In this example: `10011100'

How do we get an integer from a binary string? First, we know it is a string, so the elements are '1' and '0'. For every 1 in this string, we add a power of two to the overall decimal value. The power of 2 that we add depends on the position of the 1 in the binary string. A 1 in the far right (last) position of the string adds 2**0, in the next to last position adds 2**1, in the next to the next to the last position adds 2**2, and so on. If the bit is a '1', then we add that power of 2 to the overall sum; if it is 0 we do nothing.

For example, starting the last (right most) position of '10011100'

- last bit is ‘0’, so it contributes nothing to the sum.
- next bit is ‘0’, so it contributes nothing to the sum.
- next bit is ‘1’, so it contributes 2**2 to the sum.
- next bit is ‘1’, so it contributes 2**3 to the sum.
- next bit ‘1’, so it contributes 2**4 to the sum.
- next bit is ‘0’, so it contributes nothing to the sum.
- next bit is ‘0’, so it contributes nothing to the sum.
- next bit is ‘1’, so it contributes 2**7 to the sum.

The decimal equivalent is therefore 2**2 + 2**3 + 2**4 + 2**7, which equals 156.

Question 2

Take the following Python code that stores a string:
str = ‘X-DSPAM-Confidence:**0.8475**’
Use find and string slicing to extract the portion of the string after the colon character and then use the float function to convert the extracted string into a floating point number.

Question 3

romeo.txt

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Using a text editor create the file above. Write a program to open the file romeo.txt and read it line by line. For each line, split the line into a list of words using the split function. For each word, check to see if the word is already in a list. If the word is not in the list, add it to the list. When the program completes, sort and print the resulting words in alphabetical order.

Show your code to Mr. Ngetich or me to get credit for the lab.