CSCI 101 Final Exam Review -- Solutions

1. Review all posted notes, slides, examples on the course web site. Review all homework assignments and midterms. Review posted sample solutions.

2. Lists

   a) def indexMin(t):
      minIndex = 0
      for i in range(1,len(t)):
         if t[i] < t[minIndex]:
            minIndex = i
      return minIndex

   b) [8, 5, 0, -7, 4]
      [8, 13, 0, -7, 4]
      [8, 13, 13, -7, 4]
      [8, 13, 13, 6, 4]
      [8, 13, 13, 6, 10]

3. Loops and nested loops

   a) def hailstone(n):
      while n != 1:
         print(n, end=' ')
         if n % 2 == 0:
            n = n // 2
         else:
            n = 3*n + 1
      print(n)

   b) def sumOfList(t):
      """ Returns the sum of all values in list t."""
      sum = 0
      i = 0
      while i < len(t):
         sum += t[i]
         i += 1
      return sum

   c) Double-nested loops:
      1. printSquare(5,'#') produces
         #####
         #####
         #####
         #####
         ######
2. def printTriangle(count, symbol):
    for i in range(count):
        for j in range(i+1):
            print(symbol, end='')
        print()

4. Dictionaries
   Output produced:
   spring
   Sommer
   Herbst
   winter

5. Objects
   a) self refers to the object performing the action
   b) A class specifies the design of an object, specifically how its state is represented and what
      functionality it will have. An object is a specific instance of a class. For example, a class might
      describe the characteristics and functionality of a ball, whereas a particular green ball at position
      (10, 20) with radius 9 might be a specific object, an instance of the ball class.

6. Algorithms and Complexity
   a) Insertion sort considers each element of a list in turn, and inserts it into its proper position relative
      to the already sorted values to the left of it in the list. Worst case running time of insertion sort is
      $O(n^2)$ to sort $n$ values. Merge sort runs faster, with a worst-case running time of $O(n \log n)$ to sort
      $n$ values.
   b) 11 comparisons. The list is sorted so you could use the binary search algorithm, with a running time
      of $O(\log n)$. With $\log_2 2000 < 11$, that means 11 comparisons will suffice.
   c) 45 seconds. With the algorithm having a running time of $O(n)$, the running time will increase
      linearly with the input size. If the input size triples, so will the running time.
   d) $O(n \log n)$. Split the list of length $n$ in half, recursively sort the two halves. Keep going until you have
      lists of length 1. Then merge the sorted sublists. There are $\log n$ levels, and $O(n)$ work done at each
      level, so the overall run time is $O(n \log n)$.

7. $11100111 - 25 \quad [11100111 = -(00011001) = -(2^4+2^3+2^0) = -25]$
   $\phantom{11100111} +00011111 +31$
   $00000110 + 6$

8. Draw a logic diagram that implements the XOR function (for two inputs) using only AND, OR, and NOT
   gates. [Recall that XOR is true when exactly one of its inputs are true, but not both.]
9. HMMM solution to be posted soon.

10. A recursive (and clever :-) countEven(t) function:

   ```python
def countEven(t):
      """Returns the number of even values in list t."""
      if len(t) == 0:
         return 0
      else:
         return (((t[0] % 2) + 1) % 2) + countEven(t[1:])
```

11. The explode() function using a for loop:

   ```python
def explode(s):
      """Returns list of characters in string s."""
      t = []
      for c in s:
         t += [c]
      return t
```

12. Output from mystery function.

   a) Values returned from:
      i. mystery(['a','b','c','d']): False (because 'd' is not in the input list)
      ii. mystery(['a','b','c','c']): True (because 'c' appears in the input list)

   b) The mystery function takes as input a list and a value, and determines whether the value appears in the list.

13. ```python
def drawKoch(length, levels):
    if levels == 0:
       turtle.forward(length)
    else:
       drawKoch(length/3, levels-1)
turtle.left(60)
drawKoch(length/3, levels-1)
turtle.right(120)
drawKoch(length/3, levels-1)
turtle.left(60)
drawKoch(length/3, levels-1)```