4COM2002 Introduction to Programming and Discrete Structures

Basic Python Programming Exercises 5

Practice using counted while loops

Example:

Re-code the function ***zeroToNine*** from Exercise Set 3, using a counted while loop instead of a for loop that iterates over a range.

The function has no parameters, and prints out the whole numbers 0 to 9 in ascending order

set *x* to 0
while *x* is less than *10*
    display the value of *x*
    add 1 to *x*

```
def zeroToNine () :    # Alternative 1
    x = 0
    while x < 10 :
        print (x)
        x = x + 1
```

set *x* to 0
while *x* is less than or equal to *9*
    display the value of *x*
    add 1 to *x*

```
def zeroToNine () :    # Alternative 2
    x = 0
    while x <= 9 :
        print (x)
        x = x + 1
```

1. Re-code the following functions from Exercise Set 3 so that they use counted while loops instead of for loops with ranges. Test that both versions (using while and using for) give correct results. Do as many as you need to in order to understand how to code a counted while loop

   oneToTen

   oneToN

   squaresToN

   repeatPrint

   power

   factorial

Implement the following functions, using counted while loops

2. ***countInCubes*** (*n*)

```
# This function takes a single parameter n
# It employs a while loop and the function power (defined above) to calculate and print out the
# cubes of all the whole numbers between 1 and n
    set counter to 1
    while counter is less than n+1
      set  cube  to  power(counter,3)
      display the value of  cube
      add 1 to counter
```

3. ***fibonnacci*** (*n*)

```
# This function takes a single parameter n
# It employs a while loop and several local variables to calculate and display the first n Fibonnacci
numbers
    set prevNum to 0
    set currentNum to 1
    set nextNum to 1
    set counter to 1
    while counter is less than n+1
          display currentNum
          set prevNum equal to currentNum
          set currentNum equal to nextNum
          set nextNum to prevNum + currentNum
          add 1 to counter
```

4. Write, test and debug a function called ***isPrime*** that takes a positive whole number, n, as its single parameter and *returns True* if n is a prime number and *returns False* if it is not.

   A prime number is a whole number that is divisible by no numbers other than 1 and itself
   Note that 1 is not a prime number

   Try implementing this with a while loop and with a for loop

5. Write, test and debug a function called ***printPrimes*** that takes a positive whole number, n as its single parameter and prints out all of the prime numbers between 1 and n inclusive. Use isPrime to determine whether or not a number is a prime number.

   Try implementing this with a while loop and with a for loop

6. Write, test and debug a function called ***printFactors*** that takes a positive whole number, n as its single parameter and prints out all of its factors, including 1 and n.

   A number, x, is a factor of another number, n, if n is divisible by x
   Try implementing this with a while loop and with a for loop

7. Write, test and debug a function called ***printPrimeFactors*** that takes a positive whole number, n as its single parameter and prints out all of its prime factors.

   Try implementing this with a while loop and with a for loop

Practice using an uncounted (but definite) while loop

8. Write, test and debug a function called ***gcd*** that employs Euclid's algorithm for finding the Greatest Common Divisor (GCD) of two numbers.

   This function takes two positive numbers, a and b, as its parameters and returns the GCD of the two (the largest whole number that is a divisor of both a and b). See notes from Strand A (Discrete Structures) on Euclid's algorithm for finding the GCD of two numbers

Practice using indefinite while loops (make sure you attempt these)

9. Write, test and debug a Python function without parameters called ***mean***, which collects a series of numbers from a user, and print out the arithmetic mean of those numbers

   The easiest way to do this is to create a function definition that encapsulates the code given in this week's programming lecture (Strand B, Lecture 6)

   Try a couple of different versions of the code

10. Write, test and debug a function without parameters called ***positives*** which collects a series of numbers from a user (like you did when calculating the *mean*), and prints out how many of them are positive (greater than 0)

11. Write, test and debug a function without parameters called ***signs*** which is a modified version of *positives* that prints out how many of the numbers are positive (greater than 0), how many are negative (less than 0), and how many are 0

12. Write, test and debug a function without parameters called ***greatest*** which collects a series of numbers from a user (like you did when calculating the *mean*), and prints out the number that is the greatest

13. Write, test and debug a function without parameters called ***longestOfAll*** which collects a series of strings from a user and prints out the string that is the longest. If two or more strings are of the longest length the function should print the one that the user typed in first.

More challenging exercises

14. Write, test and debug a function ***daysBetween*** that returns the number of days between two dates

    The function has six parameters, representing two dates, as follows d1,m1,y1,d2,m2,y2

    where d1,m1,y1 are three whole numbers representing the first date, in the format <day number>, <month number>, <year number>

    and d2,m2,y2 are three whole numbers representing the second date, in the format day number, month number, year number

    If the first date is earlier than the second date the function should return a positive number

    If the first date is later than the second date the function should return a negative number

    If the two dates are the same the function should return zero

    You may assume that only legal dates (starting from 1st january in the year 0 CE) will be given as arguments when the function is called, so your function does not have to deal with parameters that do not represent real dates.

    You will find that functions such as isLeapYear, daysInYear and daysInMonth that you worked on in earlier exercises will be useful in the solution of this problem.

15. Write, test and debug a function ***squareRoot*** that takes a non-negative number, *n*, as its single parameter and returns the square root of that number that is within 0.01% of the true square root of *n*. You should use Newton's method as described in Chapter 10 of Think Python

16. Write, test and debug a function without parameters called ***graphics***. This function should present a list of options for the user to choose from, each of which employs one of the drawing functions you wrote for Exercise Set 3 (I suggest you start off with two or three). The user should indicate which one they want to choose by entering a number,