# Loop Exercises Set 3

1. a) A prime number is an integer greater than 1 that is evenly divisible by only 1 and itself. For example, 2 ,3 ,5 ,7 are prime numbers, but 4, 6, 8 and 9 are not. Create a PrimeNumber application that prompts the user for a number and then displays a message indicating whether the number is prime or not. Hint: The % operator can be used to determine if one number is evenly divisible by another.
2. 
   b) Modify the application to prompt the user for two numbers and then display the prime numbers between those numbers.

3. The Fundamental Theorem of Arithmetic states that every positive integer is the product of a set of prime numbers. This is the prime factors. For example, the prime factors for 140 are 2, 2, 5, and 7 (2*2*5*7=140). Create a PrimeFactors application that prompts the user for a positive integer and then displays that integer's prime factors.

4. Create a DigitsDisplay application that prompts the user for a non-negative integer and then displays each digit in reverse order on a separate line. Application should look similar to"

```
Enter a positive integer: 789
9
8
7
```

5. a) Create a CubeSum application that prompts the user for a non-negative integer and then displays the sum of the cubes of the digits. Application output should look similar to:

```
Enter a positive integer: 223
The sum of the cubes of the digits is: 43
```

   b) Modify the application to determine what integers of two, three, and four digits are equal to the sum of the cubes of their digits.

   Example: $371 = 3^3+7^3+1^3$

6. Create a GCD application that prompts the user for two non-negative integers and then displays the greatest common divisor (GCD) of the two numbers. The GCD is the largest integer that divides into both numbers evenly.

   The application output should look similar to:

```
Enter a number: 32
Enter a second number: 40
The GCD is: 8
```

7. An interesting (yet unsolved) question in mathematics is called "hailstone numbers." This series is produced by taking an initial integer, and if the number is even, dividing it by 2. If the number is odd, multiply it by 3 and add 1. This process is then repeated. For example, an initial number of 10 produces:

   10, 5, 16, 8, 4, 2, 1, 4, 2, 1 …

   An initial number 23 produces:

   23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, …

   Note that both numbers eventually reach the 4, 2, 1, 4, 2, 1 … cycle. Create two applications (HailStone1 and HailStone2) that answer the following questions for the initial values of 1 to 200:

   a) Do all integers for 1 to 200 eventually reach this cycle?

   b) What is the maximum number of iterations to reach the cycle and which starting number produces this maximum?

   If the integers were for 1 to 60 the output would be:

```
After inspecting the Hailstone numbers from 1 to 60,
The maximum iterations to 4,2,1 were: 112
The maximum iterations occurred at: 54 55
```