Problem Solving Methodology in IT (COMP1001)

Assignment Three
(Due at noon on 11 October 2018)

(Instructions: see the instructions in Blackboard)

Rocky K. C. Chang

30 September 2018

1. [Weight = 4] (Bitwise left shifting) A very useful function for fast multiplication and division is bit shifting. The usage of Python's Bitwise Left Shift is given in https://python-reference.readthedocs.io/en/latest/docs/operators/bitwise_left_shift.html. Write a function for this Bitwise Left Shift operator. The function signature is given below.

```
Function bitLeftShift(binIn, n):
  Input:
  - binIn: a binary number stored as a string. The ~~least~~ most
  significant bit is stored as the first character in the
  string and so forth.
  - n: the number of bits to be shifted and n >= 0.
  Output: bin(binIn << n)
```

In your .py file, you must include the following code for testing.

```
print("Test cases:")
print("Input: ", "00011, 3", "Output: ", bitLeftShift("00011", 3))
print("Input: ", "11111, 3", "Output: ", bitLeftShift("11111", 3))
print("Input: ", "00000, 3", "Output: ", bitLeftShift("00000", 3))
print("Input: ", "11111, 0", "Output: ", bitLeftShift("11111", 0))
print("Input: ", "10101, 5", "Output: ", bitLeftShift("10101", 5))
print("Input: ", "00001, 8", "Output: ", bitLeftShift("00001", 8))
```

The outputs should be

```
Test cases:
Input:  00011, 3 Output:  11000
Input:  11111, 3 Output:  11111000
Input:  00000, 3 Output:  0
Input:  11111, 0 Output:  11111
Input:  10101, 5 Output:  1010100000
Input:  00001, 8 Output:  100000000
```

You cannot use any built-in functions, such as *int()*, *eval()*, and *len()*. You also cannot use any string methods. You also should not convert it to decimal number and so on. Therefore, the remaining choice is to use a control loop to process the input string.

2. [Weight = 3] (Computing $\sqrt{2}$) $\sqrt{2}$ is a very interesting number. It has been analyzed and studied for thousands of years. One of the problems is to estimate its value. One of the methods is the Babylonian method which first picks an initial guess, $a_0 > 0$. Starting from a_0, the subsequent a_i, $i > 0$, will be computed based on

$$a_{n+1} = \frac{a_n + \frac{2}{a_n}}{2} = \frac{a_n}{2} + \frac{1}{a_n}.$$

Given $a_0 = 1$, you can easily verify that

- a_1 = ~~1/2~~ 3/2 = **1**.5
- a_2 = 17/12 = **1.41**6...
- a_3 = 577/408 = **1.41421**5...
- a_4 = 665857/470832 = **1.4142135623**746...

...

The approximation is getting more accurate as more terms are obtained. Using 65 decimal places, $\sqrt{2}$ = 1.41421356237309504880168872420969807856967187537694807317667973799.... In order to increase the accuracy, in this question we will compute the rational numbers. Write a function called ~~*sequareRootTwo()*~~ *squareRootTwo()* to compute the rational number for $\sqrt{2}$).

```
Function sequareRootTwo(n,a)
  Input: n is a non-negative integer and a is the initial
  value (i.e., a0) which is a positive integer.
  Output: a print out of a0, a1, a2, …, an in the form of
  rational number and the form of using radix points. The
  exact format is given below for n = 6.
```

```
n =  0 the number is:  1/1
n =  0 using the radix point, the number is:  1.0

n =  1 the number is:  3/2
n =  1 using the radix point, the number is:  1.5

n =  2 the number is:  17/12
n =  2 using the radix point, the number is:  1.4166666666666667

n =  3 the number is:  577/408
n =  3 using the radix point, the number is:  1.4142156862745099

n =  4 the number is:  665857/470832
n =  4 using the radix point, the number is:  1.4142135623746899

n =  5 the number is:  886731088897/627013566048
n =  5 using the radix point, the number is:  1.4142135623730951

n =  6 the number is:  1572584048032918633353217/1111984844349868137938112
n =  6 using the radix point, the number is:  1.4142135623730951
```

In your .py file, you must include the following code.

```
sequareRootTwo(10,1)
sequareRootTwo(10,10)
sequareRootTwo(10,50)
```

What are the differences in their results? Answer this in a comment at the end of your .py file.

3. [Weight = 3] (A number-guessing game) This game is played by two parties. The first one comes up a number, and the second will guess it. The first player can only reply whether the guessed number is too big/small compared with the actual number. Implement the following function for this game. You may import the random module and use *random.randint()* to generate a random number.

Function *guessNumber():*
Input: *None*
Output:
- *The user will first be asked to input the lower and upper integers of a range of integers. Both the lower and upper integers are included.*
- *The program will generate a random number from this range.*
- *The program will then ask the user for a guess.*
- *If the guess is outside the range, the program will print "out of range" and prompts the user for another input.*
- *The program will respond to the user's guess by printing "the number is too small/big."*
- *The program will exit when the user guesses it correctly and print a congratulation message with the total number of attempts.*

A sample game rounds is given below.

```
Please enter the lower and upper numbers, separated by a comma: 1,100
The range of numbers is [1...100].
Attempt #1: Please guess a number. 0
Your input is out of range.
Attempt #1: Please guess a number. 101
Your input is out of range.
Attempt #1: Please guess a number. 50
Your number is too big.
Attempt #2: Please guess a number. 25
Your number is too small.
Attempt #3: Please guess a number. 35
Your number is too big.
Attempt #4: Please guess a number. 30
Congratulation, you got it after 4 guesses.
```

Please make sure that you put *guessNumber()* as the last line in the .py file.