Assignment #2
Course: CPSC-442 (Python for Data Science)

1. Write a program to design the following structure using control statements.

   a.)

```
*
**
***
****
*****
```

   b.)

```
    *
   **
  ***
 ****
*****
```

   c.)

```
    **
   ****
  ******
 ********
**********
```

   `Hint: You may make use of zip() function. Read about it in python documentation.`

2. We add a Leap Day on February 29, almost every four years. The leap day is extra and we add it to the shortest month of the year, February.
   There `are three criteria to identify leap years`:
   1. The **year** can be evenly divided by 4;
   2. If the **year** can be evenly divided by 100, it is NOT a **leap year**, unless;
   3. The **year** is also evenly divisible by 400. Then it is a **leap year**.

   This means that in the Gregorian calendar, the years 2000 and 2400 are leap-years, while 1800, 1900, 2100, 2200, 2300 and 2500 are NOT leap-years.

   Your task is to write a program with a function to verify whether the integer is a leap year or not. The function should return True/False.

Example:

```
Input: 1900
Output: False
Input: 1992
Output: True
```

3. International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows: `"a"`maps to `".-"`, `"b"` maps to `"-..."`, `"c"` maps to `"-.-."`, and so on.

   For convenience, the full table for the 26 letters of the English alphabet is given below:

```
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.",
"---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]
```

Check this page https://morsecode.scphillips.com/morse2.html

Now, you need to write a function to return the morse code for the corresponding input word.

Note: the input word would be in lower case.

Example:

```
Input: word = "gin"
Output: '--...-.'
Input: word = "zen"
Output: '--...-.'
```

4. Given a positive integer, output its 2's complement number. The 2's complement strategy is to flip the bits of its binary representation and then add 1 to the binary representation.

   You need to write the definition for the method `find2sComplement().`

   Definition for `findComplement()` is already provided.

   Example 1:

```
Input: 5
Output: 3
```

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its co
mplement is 010. So you need to add 1 to 010 resulting to 011 i.e. 3

Example 2:

Input: 10

Output: 6

Explanation: The binary representation of 10 is 1010 (no leading zero bits), and its complement is 101. So you need to add 1 to 101 resulting to 110 i.e. 6

```
def find2sComplement(num):
        #Following method would return the 1’s complement of the decimal number.
        complement=findComplement(num)
        # write code
```

```
def findComplement(num):
        numWord=bin(num)         #converts decimal number to binary representation
        numWord=numWord[2:]      #trims initial 2 chars from the string
        length=len(numWord)      #calculates the length of the string
        i=0
        res=''
        #Following loop would iterate for every char to convert it.
        while i<length:
                if numWord[i]=='1':
                        res+='0'
                else:
                        res+='1'
                i+=1
        return res
```

5. Consider a robot at position (0, 0). Given a sequence of its moves, judge if this robot makes a circle, which means it moves back to **the original place**.

The move sequence is represented by a string. And each move is represented by a character. The valid robot moves are `R` (Right), `L`(Left), `U` (Up) and `D` (down).

Write a function to verify if the robot makes a circle. The output should be True or False.

Example 1:

```
Input: "UD"
Output: True
```

Example 2:

```
Input: " LDRRLRUULR"
Output: False
```

Example :3

```
Input: " RRDD"
Output: False
```

6. Ask the user for a number, then a word, and then print out a phrase that depends on the number and the word. You should pluralize the word by adding an "s" to the end whenever they enter a number that is 0 or greater than 1.

```
In: 0
In: bat
Out: 0 bats
In: 1
In: cat
Out: 1 cat
In: 4
In: dog
Out: 4 dogs
```

Let's implement some more advanced rules:

Words ends in:	**Becomes:**	**Example:**
-ife	-ives	life -> lives
-sh/ch	-shes/ches	bush -> bushes church -> churches
-us	-i	cactus -> cacti guy -> guys

-ay/oy/ey/uy	-ays/oys/eys/uys	toy -> toys key -> keys way -> ways
-y	-ies	fly -> flies
everything else	add -s	hat -> hats

To do this, you'll have to use string slicing. The following table provides the shortcuts for common tools you might need:

```
word = "computerz"

print(word[:5])  # prints "compu"

print(word[:-1])  # prints "computer"

print(word[4:])  # prints "uterz"

print(word[-3:])  # prints "erz"
```

7. Write a function **is_prime** that takes in one integer as an argument and determines whether it is a prime number or not. As a reminder, a prime number is only divisible by 1 and itself. **is_prime** should return **True** or **False** depending on whether the number is prime.

8. Write a function **snake_case** that takes in a string in camelCase (like you would write variables in JavaScript), and converts it to snake_case, converting everything to lowercase and separating by underscores. You can assume that each capital letter is the start of a new word. Use string slicing to get "parts" of a word.

9. Write a function **get_number_input** that takes in a string as the prompt and uses that prompt to ask the user for input using **input()**. However, in this function, you must ensure that the user enters a number (int or float); otherwise, it must ask the question again. Return the user's input only when they enter a valid number. It should be able to accept negative numbers and decimal points.