ICS Fall 2021
Assignment 1

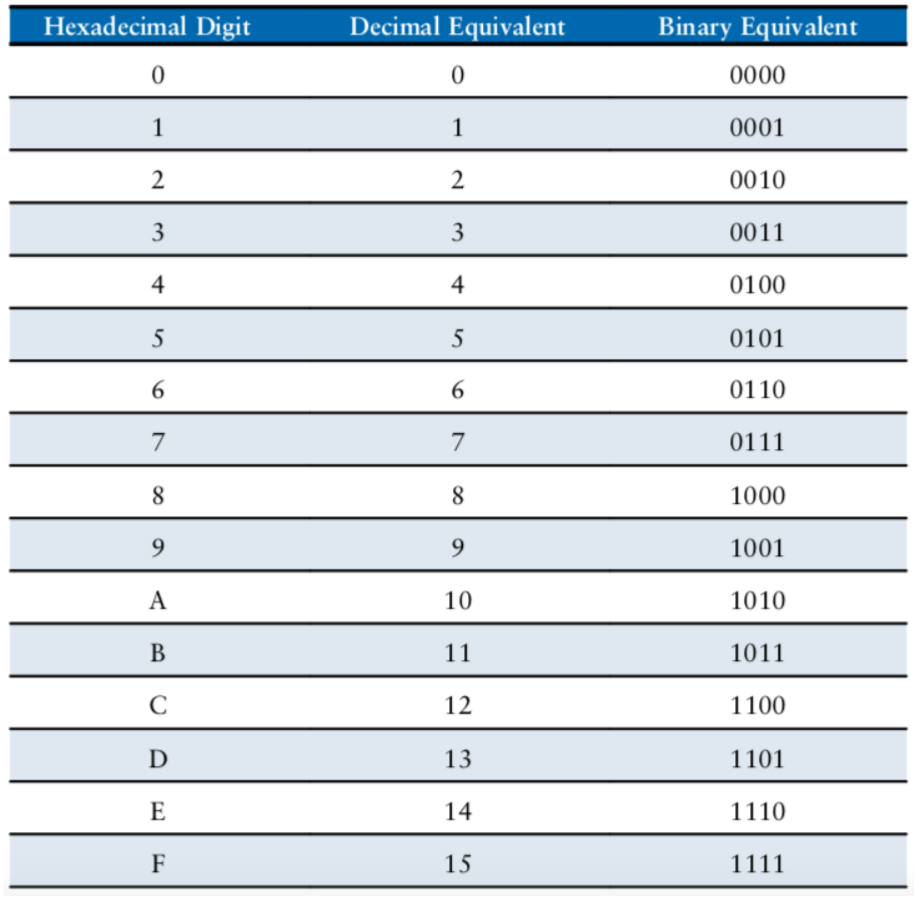
Please note: For some questions, starting codes are available. Please fill in blanks in starting codes. For questions without starting codes, you can write the solutions in any form you like. Since we use an autograder to grade on your code, please strictly name the functions and the submitted files as required by the questions. Moreover, when you submit your answers, you should zip the files, not the whole folder of your answers. Lastly, the starting codes may contain some tests for you to check if your code is correct.

1. Prime Numbers

A prime number is a number that is greater than 1 and only evenly divisible by itself and 1. For example, the number 5 is prime because it can only be evenly divisible by 1 and 5. The number 6, however, is not prime because it can be divided by 1, 2, 3, and 6. Write a function named `is_prime` which takes a positive integer as the argument and returns True if the number is prime, and False otherwise. (Note: Please put your code in `Q1_prime.py`.)

2. Hexadecimal number

In computer science, we also use hexadecimal numbers. Hexadecimal number is a number system of base 16. It uses the digits from 0 to 9 along with letters A to F. One column in a hexadecimal number represents 2^4 = 16 possibilities (i.e., half of a byte). As you know, writing long binary numbers is tedious and prone to error, so converting them into hexadecimal numbers sometimes can relieve the pain and make the work more convenient. The following table shows the map among hexadecimal, decimal and binary. Write a function that converts an integer into a hexadecimal number. (Please complete the `Q2_int2hex.py`.)



Hexadecimal Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

3. Camel case or snake_case?

A good coding behavior accepts two naming styles: the camelCase and snake_case.

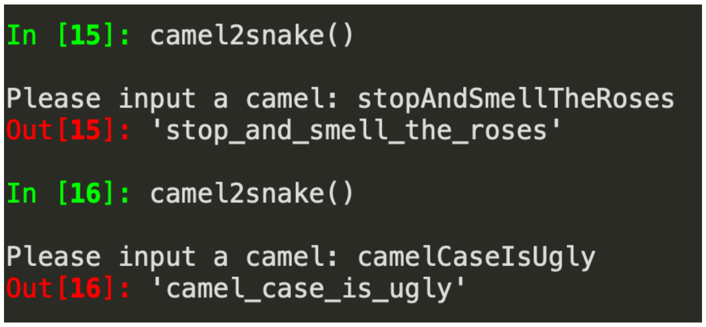
A camelCase string is a string in which all of the words are run together but the first character of each word is uppercase, except the first word. A snake_case string is a string in which the words are separated by underscores('_').

Write a function named `camel2snake` that asks the user to input a camelCase string and converts the string to a snake_case string and returns it . (complete the `Q3_camel2Snake.py`)

Examples:

input	stopAndSmellTheRoses	camelCaseIsUgly
output	stop_and_smell_the_roses	camel_case_is_ugly

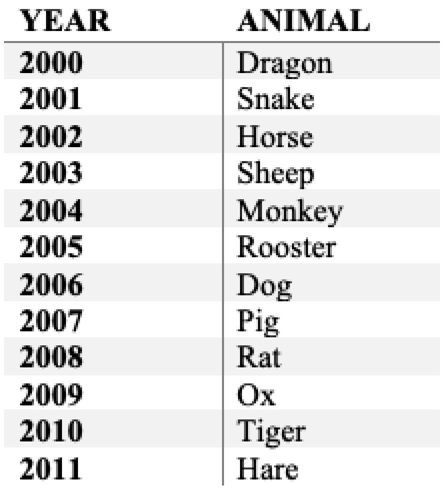
When the function is called, with the inputs mentioned in the example, the expected outputs should like the following,



```
In [15]: camel2snake()

Please input a camel: stopAndSmellTheRoses
Out[15]: 'stop_and_smell_the_roses'

In [16]: camel2snake()

Please input a camel: camelCaseIsUgly
Out[16]: 'camel_case_is_ugly'
```

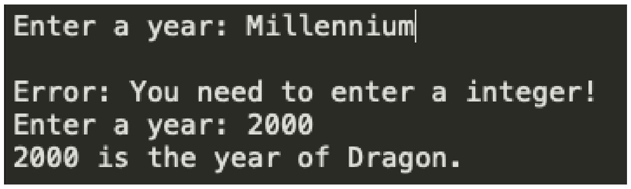
4. Chinese Zodiac

The Chinese Zodiac assigns animals to years in a 12-year cycle. One 12-year is shown in the table below. The pattern repeats from there, with 2012 being another year of the dragon, and 1999 being another of the hare.



YEAR	ANIMAL
2000	Dragon
2001	Snake
2002	Horse
2003	Sheep
2004	Monkey
2005	Rooster
2006	Dog
2007	Pig
2008	Rat
2009	Ox
2010	Tiger
2011	Hare

Write a program that reads a year from the users and displays the animal associated with that year. Your program should work correctly for any year greater than or equal to Zero, not just the ones listed above in the table. The following is an example of a kind of output of your script. (complete the `Q4_zodiac.py`)



```
Enter a year: Millennium

Error: You need to enter a integer!
Enter a year: 2000
2000 is the year of Dragon.
```

5. Does "2.07 - 2 = 0.07"? (put your answer in `Q5_roundOffError.py`)

What will you get when input "2.07 - 2" into a Python console? Actually, you won't have 0.07 but 0.06999999999999984. What happens? Well, don't worry, your machine is fine. Actually, this problem is normal when using digital circuits to represent numbers, which is called **truncation error** or **round-off error**, meaning the part of the value being stored is lost because the mantissa field is not large enough. In this question, we need to manually show how this truncation error occurs.

Let's have a look at how to convert a decimal fraction number into a binary string. This conversion can be done by the following steps:

1. multiplying the fraction by 2
2. Splitting the result into an integer and a new fraction
3. Go to step 4, if the new fraction is 0; otherwise, go to step 1
4. Concatenate all integers to form the binary, and stop.

The following is an example. Converting 0.6875 into binary:

0.6875 x 2 = **1** + 0.375
0.375 x 2 = **0** + 0.75
0.75 x 2 = **1** + 0.5
0.5 x 2 = **1** + 0

So, by concatenating the integers and adding a '0.' at the beginning, we have the binary fraction as **0.1011**.

By the way, for some numbers, the above algorithm will convert them into a binary string of infinity length with some recurring pattern, for example, 0.1 becomes 0.0001100110011. And, this is where the imprecision comes from.

Now, your first task is to complete a function: `fraction2binary( f, l)` which converts the input fraction "f" into a binary of length "l". It returns the binary fraction.
Your second task is to convert the binary string of a fraction back into decimal. Please complete the function `binary2fraction(b)`, which converts the input fraction binary `b` number into the decimal fraction.

Hint:

1. There are two ways to do this. Apart from the one we introduced in the lecture slides, you can also first convert the fraction part of the binary into decimal, then, divide it by 2**n, where n is the length of the fraction part, for example, you can calculate the decimal of a binary fraction, 0.1011, by taking 1011 as an binary integer, converting it into decimal, then divide it by 2**4.
2. You can use the built-in functions in this task such as `int()`