ICS 2024 Summer Assignment 1

Please note: For some questions, starting codes are available. Please fill blanks in the starting codes. For questions without starting codes, you can write the solutions in any form you like. Since we use an auto-grader to grade your code, please strictly name the functions and the submitted files as required by the questions. Moreover, when you submit your answers, you'd better upload all the required files separately. **Please don't zip your folder and upload it**. 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 is 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 `prime.py`.)

2. Matrix Mirror

Write a function named "`fun1`", which takes an NxN square of numbers, i.e., a 2-D list, and flips it horizontally **in place**. (put your code in the `matrix_mirror.py`)

Examples:

`m is assigned as`	`[[1, 2],`
	`[3, 4]]`
		`[[1, 2, 3],`
		`[4, 5, 6],`
		`[7, 8, 9]]`
			`[[1, 2, 3, 4],`
			`[5, 6, 7, 8],`
			`[9, 10, 11, 12],`
			`[13, 14, 15, 16]]`

`after calling fun1(m)`			

m becomes	[[2, 1], [4, 3]]	[[3, 2, 1], [6, 5, 4], [9, 8, 7],]	[[4, 3, 2, 1], [8, 7, 6, 5], [12, 11, 10, 9], [16, 15, 14, 13]]

```
input m: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
after running fun1: [[3, 2, 1], [6, 5, 4], [9, 8, 7]]
input m: [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
after running fun1: [[4, 3, 2, 1], [8, 7, 6, 5], [12, 11, 10, 9], [16, 15, 14, 13]]
```

3. Integer to the Roman letter

Write a function named `convert_to_Roman_numeral` that accepts an integer between 1 and 3999, converts it into a Roman numeral, and returns the Roman number. Here is the conversion table:

'I'	'V'	'X'	'L'	'C'	'D'	'M'
1	5	10	50	100	500	1000

We take `4 as IV, 9 as IX, 40 as XL, 90 as XC, 400 as CD, 900 as CM`, and so on.

Specifically, the numbers from 1 to 10 are expressed as Roman numerals as follows:

`I, II, III, IV, V, VI, VII, VIII, IX, X.`

The system, being basically decimal, tens, and hundreds follows the same pattern. Thus, 10 to 100 (counting in tens, with X taking the place of I, L taking the place of V, and C taking the place of X):

`X, XX, XXX, XL, L, LX, LXX, LXXX, XC, C.`

Similarly, 100 to 1000 (counting in hundreds):

`C, CC, CCC, CD, D, DC, DCC, DCCC, CM, M.`

Examples:

input	1776	1949	3999
output	MDCCLXXVI	MCMXLIX	MMMCMXCIX

Put your code in `int2Rome.py`.

Hint: What data structure will you use to connect Roman and Arabic, the `list` or the `dict`? Which one can make your code shorter?

4. Encrypting the contents

Julius Caesar protected his confidential information by encrypting it using a cipher. Caesar's cipher shifts each letter by a number of letters. If the shift takes you past the end of the alphabet list, just rotate back to the front of the list. In the case of a rotation by 3, w, x, y, and z would map to z, a, b, and c.

```
Original alphabet:      abcdefghijklmnopqrstuvwxyz
Alphabet rotated +3:    defghijklmnopqrstuvwxyzabc
```

You need to encrypt and decrypt a message using Caesar's cipher. But this time, you need to use an alphabet list (i.e., a codebook) which is a mixture of uppercase and lowercase, and the alphabets are shuffled, not in the alphabet order. The following figure shows an example of the codebook, which is a list.

```
Your codebook:
['t', 'O', 'v', 'A', 'N', 'u', 'L', 'r', 'T', 'E', 'Q', 's', 'Y', 'j', 'P', 'X'
'a', 'g', 'l', 'b', 'J', 'I', 'd', 'p', 'q', 'z', 'n', 'x', 'R', 'C', 'B', 'M',
'Z', 'e', 'G', 'S', 'K', 'i', 'y', 'w', 'o', 'h', 'f', 'V', 'k', 'F', 'H', 'm',
'U', 'D', 'W', 'c']
```

1) Write a function name `caesarEncrypt()` which takes three arguments: `message, codebook, shift`, where `message` is a string, `codebook` is a list of alphabets, and `shift` is a positive integer. The function returns an encrypted message.
2) Write a function name `caesarDecrypt()` which takes three arguments: `message, codebook, shift`. The function returns a decrypted message.

Put your code in `caesar_encryption.py`.

Notes:

- The cipher only encrypts letters; symbols remain unchanged.
- The encode/decode of a character can be done by finding and computing its indices, but also, you may also use the Python dictionary to simplify this process.

When you run the tests in the starting code, you will get the following output. (i.e., the message = "Hello Kitty!", codebook=the codebook given in the starting code, shift=3)

```
Origin: Hello Kitty!
Encoded: DKIIV woAAh!
Decoded Hello Kitty!
```

5. Hexadecimal number

In computer science, we also use hexadecimal numbers. The 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 can sometimes 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. Note: you are **not** allowed to use the built-in function hex(). (Please complete the `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

6. Does “2.07 - 2 = 0.07”? (put your answer in `round_off_error.py`)

What will you get when inputting “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 stored value 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. The following steps can do this conversion:

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. 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.

```
2.07 - 2 = 0.06999999999999984
The binary of 0.07 in your machine:
0.0001000111101011100001010001111010111000010100011110
Influence of the Truncation error
0.06999999999999984
```

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 a binary integer, converting it into decimal, and then dividing it by 2**4.
2. This time, you **can use** the built-in functions in this task such as `int()`

7. Summing up a list of numbers by using recursion (put your answer in `sum_a_list.py`)

Write a function named `sum_a_list()` that sums up a list of numbers recursively. The function has one argument which is a list of numbers and returns the sum of all the numbers in the list. Please note: you should use recursion to complete this task.

```
In [2]: sum_a_list([5, 4, 3, 2, 1])
Out[2]: 15
```

8. Search a number in a list

There is a set of integers in [1, 1000]. Write a function that takes a list and an integer as the argument, and the function will return (True, the number of loops it has completed) if the integer is in the list, and (False, the number of loops it has completed) if it is not. [Note: You are **not allowed** to use built-in methods. You can only use if/else and loops (i.e., for or while) in this question.] (Please complete the `search.py`.)

```
>>> integers = [99, 88, 77, 101, 203, 896, 555]
>>> search(integers, 101)
>>> (True, 4)
>>> search(integers, 1001)
>>> (False, 7)
```

Assume the set of integers is in the ascending order. Can you write a function that solves the problem by using at most ten loops? (Please complete the `search_with_loop_limit.py`.)

```
>>> integers = [77, 88, 99, 101, 203, 555, 896]
>>> search(integers, 101)
>>> (True, 0)
>>> search(integers, 1001)
>>> (False, 2)
```

9. Maximum Profit

The most naive strategy to earn profit in a stock market is to buy a share of a company at the time t_b and sell it at a time t_s in the future. The price difference between t_s and t_b is the profit. For example, one can buy 1 share of Apple Inc at the time t_0 when the price is $100, and sell it at the time t_3 when the price goes to $200. So, she/he will earn $(200-100) = 100 dollars. Please write a function that takes a list of the prices $[P_0, P_1, ..., P_{n-1}]$ of a stock during a duration $t=\{t_0, t_1, ..., t_{n-1}\}$ as the input. The function will compute the price differences $P_j - P_i$, where $j>i$, and return the maximum one. (Please complete the max_profit.py.)

Examples:

```
>>> prices = [800, 300, 600, 200, 100]
>>> max_profit(prices)
>>> 300
>>> prices = [800, 300, 200, 100, 0]
>>> max_profit(prices)
>>> -100
```

[Bonus: Can you solve it with only one loop?]