HOMEWORK 7

This homework is due on Thursday March 20th. You don't have to work on it during the break. We covered some challenging (but hopefully interesting) topics last week such as recursion and list comprehension. As a result, it would be helpful to look at them when you are back to campus.

1. Recursion and recursive functions

Problem 1.

(1) Write a function named recursive_sum(alist) that takes a list of numbers as input and returns the sum of the squares of all elements in the list. For example

```
alist = [1,2, -3]
recursive_sum(alist)
```

should return $1^2 + 2^2 + (-3)^2 = 14$.

(2) Write another function non_recursive_sum(alist) to achieve the same goal.

Problem 2.

(1) Write a function name recursive_sum_power_of_two(n) that takes a positive integer n as an input and returns the sum of the first n-powers of 2

$$1 + 2 + 2^2 + \ldots + 2^{n-1}.$$

For example

```
print(recursive_sum_power_of_two(4))
```

should return 15 because $2^0 + 2^1 + 2^2 + 2^3 = 15$.

(2) Write another non-recursive function to achieve the same goal.

Remark 1.1. Can you predict the general formula for the sum

$$1 + 2 + 2^2 + \ldots + 2^{n-1}.$$

Problem 3.

(1) Write a recursive function recursive_count_vowels(s) that takes a string s as input and returns the number of vowels (a, e, i, o, u) in s, ignoring case.

(2) Write a non-recursive function to achieve the same goal.

Problem 4.

(1) Write a recursive function recurive_sum_of_powers(n, k) that computes the sum of the first n integers raised to the power of k.

$$1^k + 2^k + \ldots + n^k.$$

For example

```
sum_of_powers(3, 2)
```

should return 14 because $1^2 + 2^2 + 3^2 = 14$.

(2) Write a non-recursive version to achieve the same goal.

Problem 5. An arithmetic progression is a sequence of numbers such that the difference from any succeeding term to its preceding term remains constant throughout the sequence. The constant difference is called the common difference of that arithmetic progression (often denoted by d). For example

$$1, 4, 7, 10, 13, \ldots$$

is an arithmetic progression with $d = 3$. An arithmetic sequence can be defined recursively as follows

$$a_0 = a, a_n = a_{n-1} + d.$$

In the above example, $a = 1, d = 3$.

(1) Write a recursive function arithmetic_sequence(a, d, n) that takes the first team (a), the difference (d), and the index (n) as input and returns the n-th term of the corresponding arithmetic sequence. For example

```
arithmetic_sequence(1, 3, 4)
```

should return 13.

(2) Write a non-recursive function to achieve the same goal. **Hint:** You can create a sequence $[a_0, a_1, \ldots, a_{n-1}]$ using the append method (creating this list is not absolutely necessary but you might find it more intuitive.)

Problem 6. The Lucas sequence is defined as follows, $L_0 = 2, L_1 = 1$ and for $n \geq 2$

$$L_n = L_{n-1} + L_{n-2}.$$

The first few terms of this sequence are

$$2, 1, 3, 4, 7, 11, \ldots$$

(1) Write a recursive function named lucas(n) that returns that n-th Lucas number.

(2) Write a non-recursive version to achieve the same goal.

So far, we have learned how recursion works for linear recursion. The same principle works for non-linear recursive relations as well.

Problem 7. Let a_n be the sequence defined as follows

$$a_0 = 1, a_n = a_{n-1}^2 + 1.$$

The first few terms of this sequence are

$$1, 2, 5, 26, 677 \dots$$

(1) Write a recursive function named sequence_a(n) that returns that n-th term of this sequence.
(2) Write a non-recursive version to achieve the same goal.

Problem 8. Let a_n be the sequence defined as follows: $b_0 = b_1 = 1$ and

$$b_n = \frac{b_{n-1}^2 + 2}{b_{n-2}}.$$

The first few terms of this sequence are

$$1, 1, 3, 11, 41, 153.$$

(1) Write a recursive function named sequence_b(n) that returns that n-th term of this sequence.
(2) Write a non-recursive version to achieve the same goal.

2. List comprehension

Do the following problems using list comprehension.

Problem 9.

(1) Given a list of numbers, create a new list containing the squares of these numbers. For example

```
numbers = [1, 2, 3, 4, 5]
# Output: [1, 4, 9, 16, 25]
```

(2) From a list of integers, create a new list that contains only the even numbers. For example

```
numbers = [1, 2, 11, 4, 9, 6]
# Output: [2, 4, 6]
```

(3) Given a list of strings, generate a new list where all the strings are in uppercase. For example

```
words = ["hello", "world", "python"]
# Output: ["HELLO", "WORLD", "PYTHON"]
```

(4) From a list of strings, create a new list that contains the first letter of each string. For example

```
words = ["apple", "banana", "cherry"]
# Output: ["a", "b", "c"]
```

Problem 10.

(1) Given a list of prices, create a new list that includes prices greater than $20. For example

```
prices = [10, 25, 30, 5, 15]
# Output: [25, 30]
```

(2) Given a string, create a new list containing only the lowercase letters. For example

```
mixed_string = "HelloWorld"
# Output: ["e", "l", "l", "o", "o"]
```

(3) Given a list of integers, create a new list that includes only the negative numbers. For example

```
numbers = [-10, 15, 0, -5, 20]
# Output: [-10, -5]
```

(4) Given a list of strings, create a new list that contains only the strings that contain the letter `e`. For example

```
words = ["apple", "banana", "cherry", "date", "fig"]
# Output: ["apple", "cherry", "date"]
```

For this problem, you might need to use the `in` operator.