CS2020
Introduction to Programming
Practice Exercise Set 3

List Processing with Loops and Functions

This exercise set is divided into two parts. In Part 1, you write code fragments to process lists. And, in Part 2, you write functions that accept and/or return lists. You will be ask to implement the code using three techniques: direct access, indexing access with the for loop, and indexing access with the while loop. It should not take more than 10 minutes to derive a solution using one technique.

In this set, you will practice only one-level lists that no nested structure. Here you will deal mainly with lists of numbers (ints, floats) and lists of strings.

Do not use map/filter operations or list comprehensions in this set. There is another exercise set for asking you to practice these techniques.

You may use list methods such as find, index, count, and others, unless indicated otherwise.

Practice Topic

- Processing lists without using list methods
- Processing lists using list methods
- Processing lists with functions

Prerequisite

You need to possess knowledge on

- Basic operations such as assignments and expressions
- Input and output (print)
- Data types int, float, and str.
- Simple selection control using if-else
- Basic looping (Practice Exercise Set 1)
- Defining and using functions (Practice Exercise Set 2)

Example 1

L is a list of numbers. Find the sum of the numbers without using any list methods or the built-in function sum.

a. Direct Access Loop

```
total = 0
```

```
for num in L:
    total += num
```

b. Indexing Access with for

```
total = 0
for idx in range(len(L)):
    total += L[idx]
```

c. Indexing Access with while

```
total, idx = 0, 0
while idx < len(L):
    total += L[idx]
    idx += 1
```

Example 2

Write a function get_total that accepts one parameter L—a list of numbers. Return the sum of the elements in L without using any list methods or the built-in function sum.

a. Direct Access Loop

```
def get_total(L):
    total = 0
    for num in L:
        total += num
    return total
```

b. Indexing Access with for

```
def get_total(L):
    total = 0
    for idx in range(len(L)):
        total += L[idx]
    return total
```

c. Indexing Access with while

```
def get_total(L):
    total, idx = 0, 0
    while idx < len(L):
        total += L[idx]
        idx += 1
```

```
    return total
```

Part 1 Code Fragments

For each question, provide a solution using direct access, indexing with the for loop, and indexing with the while loop.

1. Create a list of integers by inputting values from the user. Assume the user will enter valid values only. Stop the repetition when the user enters 0.
2. Create two lists of integers by inputting values from the user. The first list contains negative integers and the second list positive integers. Assume the user will enter valid values only. Stop the repetition when the user enters 0.
3. Given a list of integers L, find the maximum and minimum without using the built-in max and min functions.
4. Given a list of floats numbers, find out how many are less than or equal to the average of all elements.
5. Given a list of floats numbers, create two lists. The first list contain those elements that are less than or equal to the average and the second list those that are greater than the average.
6. Given a list of strings words, determine how many strings in the list have length 5 or less.
7. Given a list of strings words, determine the average length of the strings in the list.
8. Input two integers low and high, create a list of all integers between low and high, inclusive. Assume low is less than or equal to high.
9. Input two integers low and high, create a list of all odd integers between low and high, inclusive. Assume low is less than or equal to high. Don't include low or high if they are even.
10. Given a list of integers L and an integer N, determine how many times N occurs in L. Do not use the count method.
11. Given a list of integers L and an integer N, create a list of index positions where N is found in L. If there is no N in L, then the result is an empty list.
12. Without using any built-in functions, create a reversed copy of a list L.
13. Given a list of items L and an item X, create two lists. Assume X is unique in L and idx is the position that X is found in the list. The first list contains L[0], … , L[idx] and the second list the remaining items in L.
14. Given a list of strings words, create a single string by concatenating strings in the words list separated by one blank space.
15. Given a list of strings words, determine if any string in the words list contains an uppercase letter. Print out a boolean value True if there's a string with an uppercase letter and False otherwise.

Part 2 Functions

For each question, provide three versions of a function definition with a) direct access looping, b) indexing access with for, and c) indexing access with while.

1. Write a function get_int_list that returns a list of integers entered by the user. Assume the user will enter valid values only. Stop the repetition when the user enters 0.

2. Write a function get_two_int_lists that returns two lists of integers entered the user. The first list contains negative integers and the second list positive integers. Assume the user will enter valid values only. Stop the repetition when the user enters 0.
3. Write a function get_min_max that accepts a list of integers L and returns the maximum and minimum without using the built-in max and min functions.
4. Write a function find_smaller that accepts a list of floats numbers and returns how many are less than or equal to the average of all elements.
5. Write a function partition that accepts a list of floats numbers and returns two lists. The first list contain those elements that are less than or equal to the average and the second list those that are greater than the average.
6. Write a function get_words_of_length that accepts a list of strings words and an integer N. Return a list of words with length less than or equal to N.
7. Write a function get_average_length that accepts a list of strings words and returns the average length of the strings in the list.
8. Write a function generate_ints that accepts two integers low and high and returns a list of all integers between low and high, inclusive. Assume low is less than or equal to high.
9. Write a function generate_odds that accepts two integers low and high and returns a list of all odd integers between low and high, inclusive. Assume low is less than or equal to high. Don't include low or high if they are even.
10. Write a function count a that accepts a list of integers L and an integer N and return how many times N occurs in L. Do not use the count method.
11. Write a function find_positions that accepts a list of integers L and an integer N and return a list of index positions where N is found in L. If there is no N in L, then the result is an empty list.
12. Write a function get_reverse that accepts a list of items L and returns a reversed copy of L. Do not use any built-in functions or list methods.
13. Write a function divide that accepts a list of items L and an item X and returns two lists. Assume X is unique in L and idx is the position that X is found in the list. The first list contains L[0], … , L[idx] and the second list the remaining items in L.
14. Write a function concat that accepts a list of strings words and returns a single string by concatenating strings in the words list separated by one blank space.
15. Write a function has_upper that accepts a list of strings words and returns True if any string in the list contains an uppercase Return False otherwise.