Assignment 2 – Iterations and List Processing

Required Files

- `Assignment_2_template.py`

Note about grading

For each task, submit to Coursemology both your comments and the function definitions.

Each task will be graded based on correctness of the code. Different components in different tasks may have different weightages.

Do note that hard coding will be penalized. Unnecessary imports and redundant code may be penalized.

The above instructions should stand for all future assignments unless otherwise stated.

Task 1: Checksum for Singapore car license plate number [Total: 70 marks]

A Singapore car license plate, such as SHA402Z, E43A, etc., consists of the following:

- Prefix consisting of alphabets.
  - E.g. SHA, SGJ, QX, FBB, E
- Numerical series from 1 – 9999, without leading zeros
  - E.g. 10, 402, 2222
- Checksum letter, one of these 19 letters: (A, Z, Y, X, U, T, S, R, P, M, L, K, J, H, G, E, D, C, B)

The checksum letter of a license plate validates if that license plate is valid or not. In this task, write a program to calculate the checksum letter given a Singapore vehicle license plate.

Here is the formula for calculating the checksum letter[1]:

1. The last 2 letters of the prefix are converted into digits, where A = 1, B = 2, …, and Z = 26. If there is only 1 letter, then convert that letter into digit, and add a digit 0 to the front.
   - E.g. SBS = 2 and 19, SH = 19 and 8, E = 0 and 5

2. Add leading zeros to the numerical series to make it a 4 digit figure.
   - E.g. 10 -> 0010, 402 -> 0402, 2222 -> 2222 (no change)

3. Combine the prefix digits and the numerical series to form a series of six numbers.
   - E.g. SBS10 -> [2, 19, 0, 0, 1, 0]

4. Each individual numbers are then multiplied by the corresponding position of the six fixed numbers (9, 4, 5, 4, 3, 2)
   - E.g. SBS10 -> [2, 19, 0, 0, 1, 0] * [9, 4, 5, 4, 3, 2] = [18, 76, 0, 0, 3, 0]

5. The multiplied numbers are then added up

[1] Formula adapted from: https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Singapore#Checksum

- E.g. (18, 76, 0, 0, 3, 0) -> 97

6. Divide the sum by 19 and get the remainder.
   - E.g. 97 / 19 = 5 remainder 2

7. The remainder corresponds to one of the 19 letters (A, Z, Y, X, U, T, S, R, P, M, L, K, J, H, G, E, D, C, B) where remainder 0 corresponds to 'A', remainder 1 corresponds to 'Z' and so on.
   - E.g. Checksum letter for remainder 2 = 'Y'.

Task 1a [10 marks]

Write a function `alphabet_to_digit(c)`. This function takes in an alphabet and converts it into its digit representation where A = 1 and Z = 26. Your function should be case insensitive, ie., to handle both upper and lower case alphabets.

Note: You are strongly encouraged to use comments to describe your code where necessary.

Task 1b [10 marks]

Write a function `convert_prefix(prefix)`. This function takes in a non-empty alphabet string called `prefix` and returns a list of two integers, which are the converted digits of the last two alphabets in `prefix`. For our question, we do not limit the number of alphabets in a prefix. i.e. The prefix can be "SBSBCSBSDBSZBS", but still only the last two characters are taken for conversion.

- i.e. 'SBSBCDZBS' should return [2, 19] and 'E' should return [0, 5]

Note: You are strongly encouraged to use comments to describe your code where necessary.

Task 1c [10 marks]

Write a function `convert_num_series(num_series)`. This function takes in a non-empty string of at most four digits (a number from the numerical series) and returns a list of integers representing exactly 4 digits.

- i.e. '10' should return [0, 0, 1, 0] and '2222' should return [2, 2, 2, 2]

Note: You are strongly encouraged to use comments to describe your code where necessary.

Task 1d [10 marks]

Write a function `multiply_and_add(series)`. This function takes in a list of six integers and multiplies each integer with the corresponding number in [9, 4, 5, 4, 3, 2] to obtain a new list of six integers; it then returns the sum of these integers.

Note: You are strongly encouraged to use comments to describe your code where necessary.

Task 1e [10 marks]

Task 1e: Write the function `remainder_to_checksum_letter(remainder)` that takes in an integer checksum `remainder` (ranging from 0 to 18) obtained in step 6 of the formula and returns the corresponding checksum letter (in uppercase).

- i.e. 0 should return 'A' and 18 should return 'B'.

Note: You are strongly encouraged to use comments to describe your code where necessary.

Task 1f [20 marks]

Complete the function `checksum_calculator(plate)` by putting everything together and returning the checksum letter in uppercase given a license plate. Reminder: 1. Your program should be able to handle lowercase inputs. 2. The prefix can be arbitrarily long.

- i.e. 'SBS10' should return 'Y', 'sbs10' should also return 'Y'

Note: You are strongly encouraged to use comments to describe your code where necessary.

Task 2: Calculating the Value of a Polynomial [30 marks]

A polynomial is a mathematical expression that can be built using constants and variables by means of addition, multiplication and exponentiation to a non-negative integer power. While there can be complex polynomials with multiple variables, in this exercise we limit our scope to polynomials with a single variable.

A polynomial function P(x) in a single variable x takes the following general form:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

where $a_0, a_1, \ldots, a_n$ are the constants (formally called "coefficients") and x is the variable. The degree of a polynomial is the largest degree of any one term with a non-zero coefficient.

Example:

$$P'(x) = 2x^3 + x^2 - 9x + 3$$

P' is a third-degree polynomial function. We can substitute the variable x with an arbitrary value to compute the corresponding value of *P'*. For example, substituting $x = 2$,

$$P'(2) = 2 * 2^3 + 2^2 - 9 * 2 + 3 = 5$$

We can represent the constants of the polynomial function using numbers in a list. Using the example $P'(x) = 2x^3 + x^2 - 9x + 3$, we can represent the constants by [3, -9, 1, 2] in increasing degrees where 3 corresponds to the 0^{th} degree, -9 to the 1^{st} degree, 1 to the 2^{nd} degree and 2 to the 3^{rd} degree. Hence, from the number of integers in the list, we can tell what is the largest degree term of the function.

If we happen to have a term with 0 as its coefficient, we can simply do not represent it in our list. E.g. $P'(x) = 2x^3 + x^2 + 3$ can be represented by [3, 0, 1, 2]

Your Task

Your task is to write a program that can compute the value of a single variable polynomial function given the value of its single variable.

Specifically, write a function `calc_poly(const_seq,var_poly)` to compute the value of a polynomial function. The result should be rounded off to 2 decimal places whenever applicable.

The details of input arguments are as follows:

`const_seq` : a list of $< a_0, a_1, \ldots, a_i, \ldots, a_n >$ where $a_i \in \mathbb{R}$ denotes a real-numbered constant associated with the term with degree $i \in [0, n] \cap \mathbb{Z}$ and n is the degree of the polynomial.

E.g. [3, -9, 1, 2] from the example above.

`var_poly : the value of variable (x in above definition) for which the polynomial function should be evaluated.`

`var_poly ∈ ℝ is a real number.`

Please note that the symbols in the above definition follows the standard mathematical set notation. $\mathbb{R}$ is the set of all real numbers and $\mathbb{Z}$ is the set of all integers including zero.

Your function must <u>return</u> the computed value of the polynomial function.

Example:

Suppose we want to evaluate the above illustrated P'(x) for x = 2. Then, the two input arguments to `calc_poly` would be as follows:

```
const_seq = [3,-9,1,2]
```

```
var_poly = 2
```

Calling the function should return **`5`**.

Hint: Look at the equation from the lowest degree to the highest i.e. $3 - 9x + x^2 + 2x^3$
So, $P'(2) = (3 * 2^0) + (-9 * 2^1) + (1 * 2^2) + (2 * 2^3) = 5$

<u>Note: You are strongly encouraged to use comments to describe your code where necessary.</u>

Task 3: Maximum Sub-sequence Sum [Optional]

No mark will be given for solving this task, even though some marks are allotted to each sub-task. This is for your own practice.

Given a list of n non-zero integer values, a sub-sequence is defined as any non-empty sequence of at most n values that appears contiguously (i.e. next to each other) within the given list.

For example, the list [-3 4 -1 2] has the following sub-sequences:

Sub-sequence	Sum of sub-sequence
[-3 4 -1 2]	2
[-3 4 -1]	0
[4 -1 2]	5
[-3 4]	1
[4 -1]	3

[-1 2]	1
[-3]	-3
[4]	4
[-1]	-1
[2]	2

Values within a sub-sequence can be added to give the sum as shown in the above table. Moreover, the maximum sum of sub-sequences is 5 (from the sub-sequence [4 -1 2]).

In this task, you are to write a program to find the **maximum sum of sub-sequences** of a list.

- Assume there is at least one value in the list;
- Each non-zero value in the list ranges between -9 to 9 (excluding 0, of course).

This task is divided into several levels. Read through all the levels (from first to last, then from last to first) to see how the different levels are related. You are advised to start from the first level.

IMPORTANT: For Tasks 3a - 3d, please DO NOT submit your program together with the test cases provided. You should submit the function only. Failure to do so may result in failing test cases on Coursemology (and marks deduction. If you are failing test cases on Coursemology, try deleting your test cases and re-submiting again.

Within each level, sample runs are provided for you to test your program. These sample runs represent the corner test cases for small inputs. You are encouraged to expand on these and come up with your own test cases.

If you want to go directly to the main idea behind how to solve this question, you can read task 3c.

Task 3a [10 marks]:

Write the function `sum1(lst)`. The function takes in a list of integers and prints out each value on a separate line.

In addition, alongside each i^{th} value, output the cumulative sum (i.e. the sum of all values from the first value up till and including the i^{th} one).

Note: Please use the function `print` to print out your result. Refer to the template for more details. Note that every printing ends with a "#" symbol; this helps to avoid the mistake of having invisible whitespace appended to the end of your printed text. On another note, you are strongly encouraged to use comments to describe your code where necessary.

Example runs:

```
>>> sum1([1,2,3])
1,1#
2,3#
3,6#
```

```
>>> sum1([1])
1,1#
```

```
>>> sum1([-1])
-1,-1#
```

```
>>> sum1([-1,2,-3])
-1,-1#
2,1#
-3,-2#
```

```
>>> sum1([-2,1,-3,4,-1,2,1,-5,4])
-2,-2#
1,-1#
-3,-4#
4,0#
-1,-1#
2,1#
1,2#
-5,-3#
4,1#
```

Task 3b [10 marks]:

You are to modify your solution for the function `sum1` from Task 3a. Rename your function to `sum2`. The function `sum2` still takes in a list of integers. Apart from printing the output required by Task 3a, you are to print the maximum of all cumulative sums on the last line.

Note: Please use the function `print` to print out your result. Refer to the template for more details. Note that every printing ends with a "#" symbol; this helps to avoid the mistake of having invisible whitespace appended to the end of your printed text. On another note, you are strongly encouraged to use comments to describe your code where necessary.

Example runs:

```
>>> sum2([1,2,3])
1,1#
2,3#
3,6#
6#
```

```
>>> sum2([1])
1,1#
1#
```

```
>>> sum2([-1])
-1,-1#
-1#
```

```
>>> sum2([-1,2,-3])
-1,-1#
2,1#
-3,-2#
1#
```

```
>>> sum2([-2,1,-3,4,-1,2,1,-5,4])
-2,-2#
1,-1#
-3,-4#
4,0#
-1,-1#
2,1#
1,2#
-5,-3#
4,1#
2#
```

Task 3c [10 marks]:

The intuition behind finding the maximum sum of sub-sequences is simple.

If the sum of the sub-sequence from [x_i ... x_{j-1}] is negative, then we will be better off checking a new sub-sequence starting with x_j as [x_j... x_k] has a larger sum than [x_i ... x_{j-1} x_j... x_k].

For example, if we have a sequence of [1 -2 3 4], the sum of the sub-sequence [1 -2] is -1 which is negative. Hence, there will definitely be another sub-sequence sum (e.g [3] or [3 4]) that will be larger than -1. Thus, there is no need to continue checking sub-sequences that include both 1 and -2 together. Hence, we will be better off checking a new sub-sequence starting from the next number, 3.

Again, modify your solution from Task 3b. Write the program `sum3` that takes in a list of integers and outputs each value on a separate line.

Alongside each value, output the cumulative sum taking into account the restart (of checking a new sub-sequence). What this means is that if the current cumulative sum is negative, you should follow the formula above and restart the cumulative sum for the new sub-sequence.

The last line of output is the maximum over all cumulative sums with restart, which is also the maximum sum of sub-sequences.

Note: Please use the function `print` to print out your result. Refer to the template for more details. Note that every printing ends with a "#" symbol; this helps to avoid the mistake of having invisible whitespace appended to the end of your printed text. On another note, you are strongly encouraged to use comments to describe your code where necessary.

Example runs:

```
>>> sum3([1])
1,1#
1#
```

```
>>> sum3([-1])
-1,-1#
-1#
```

```
>>> sum3([1,1])
1,1#
1,2#
2#
```

```
>>> sum3([1,-1])
1,1#
-1,0#
1#
```

```
>>> sum3([-1,1])
-1,-1#
1,1#
1#
```

```
>>> sum3([-1,-1])
-1,-1#
-1,-1#
-1#
```

```
>>> sum3([1,1,-1])
1,1#
1,2#
-1,1#
2#
```

```
>>> sum3([1,-1,1])
1,1#
-1,0#
1,1#
1#
```

```
>>> sum3([-1,1,-1])
-1,-1#
1,1#
-1,0#
1#
```

```
>>> sum3([1,-1, 2])
1,1#
-1,0#
```

```
2,2#
2#
```

```
>>> sum3([-1,2,-3,4])
-1,-1#
2,2#
-3,-1#
4,4#
4#
```

```
>>> sum3([1,1,-2,2])
1,1#
1,2#
-2,0#
2,2#
2#
```

```
>>> sum3([2,-1,1])
2,2#
-1,1#
1,2#
2#
```

```
>>> sum3([-2,1,-3,4,-1,2,1,-5,4])
-2,-2#
1,1#
-3,-2#
4,4#
-1,3#
2,5#
1,6#
-5,1#
4,5#
6#
```

Task 3d [15 marks]:

Your final task is to find the starting and ending positions where the **maximum sub-sequence sum** is found. For example, in the list of values [-3 4 -1 2], the maximum sum 5 is associated with the sub-sequence between the second and fourth values [4 -1 2]. Hence the corresponding output is 5 [2,4].

In addition, if there are multiple sub-sequences with the same maximum sum, choose the one with the **minimum ending position**; if there are still multiple sub-sequences with the same minimum ending position, choose **the shortest sub-sequence**. As an example, the list [1 -1 2 -2 2 0] has a maximum sum of sub-sequences of 2 given by three sequences:

1. [1 -1 2] (within positions 1 to 3)
2. [2] (at position 3)
3. [2] (at position 5)

Choosing the minimum ending position would eliminate option (3). And between options (1) and (2) that ends at the same ending position, we choose the shorter one. Hence, option (2) is the final choice.

Again, modify your solution from Task 3c for this task. Rename your function to **sum4**.

Hint:

- *Take note of the starting positions of each sub-sequence;*
- *Among these starting positions, one of them will be the start of the sub-sequence having the maximum sum;*
- *Setting the end position is trivial.*

Note: Please use the function `print` to print out your result. Refer to the template for more details. Note that every printing ends with a "#" symbol; this helps to avoid the mistake of having invisible whitespace appended to the end of your printed text. On another note, you are strongly encouraged to use comments to describe your code where necessary.

Example runs: (Input is underlined)

```
>>> sum4([1])
1,1#
1[1,1] #
```

```
>>> sum4([-1])
-1,-1#
-1[1,1]#
```

```
>>> sum4([1,1])
1,1#
1,2#
2[1,2]#
```

```
>>> sum4([1,-1])
1,1#
-1,0#
1[1,1]#
```

```
>>> sum4([-1,1])
-1,-1#
1,1#
1[2,2]#
```

```
>>> sum4([-1,-1])
-1,-1#
-1,-1#
-1[1,1]#
```

```
>>> sum4([1,1,-1])
1,1#
1,2#
-1,1#
2[1,2]#
```

```
>>> sum4([1,-1,1])
1,1#
```

```
-1,0#
1,1#
1[1,1]#
```

```
>>> sum4([-1,1,-1])
-1,-1#
1,1#
-1,0#
1[2,2]#
```

```
>>> sum4([1,-1, 2])
1,1#
-1,0#
2,2#
2[3,3]#
```

```
>>> sum4([-1,2,-3,4])
-1,-1#
2,2#
-3,- 1#
4,4#
4[4,4]#
```

```
>>> sum4([1,1,-2,2])
1,1#
1,2#
-2,0#
2,2#
2[1,2]#
```

```
>>> sum4([2,-1,1])
2,2#
-1,1#
1,2#
2[1,1]#
```

```
>>> sum4([1,-1,2,-2,2])
1,1#
-1,0#
2,2#
-2,0#
2,2#
2[3,3]#
```

```
>>> sum4([-2,1,-3,4,-1,2,1,-5,4])
-2,-2#
1,1#
-3,-2#
4,4#
-1,3#
2,5#
1,6#
-5,1#
4,5#
6[4,7]#
```