Lab Test 2 [25 marks]

Coding time: 1 hour and 30 mins

Instructions on how to download the resource files:

1. Before the test begins, you will be instructed to download the resource files from eLearn LMS.

General Instructions:

1. You will take the lab test on your personal laptop.
2. You are not allowed to communicate with anyone or accessing any website (except for downloading/uploading files from/to eLearn) during the test. **Anyone caught communicating with another person or accessing any website during the test WILL be considered as cheating and subjected to disciplinary actions.**
3. You may refer to any file on your laptop during the test.
4. Make sure your code can generate exactly the same output as we show in the sample runs. You may be penalized for missing spaces, missing punctuation marks, misspellings, etc. in the output. You may be penalized for generating extra output.
5. Do not hardcode. We will use different test cases to test and grade your solutions.
6. Python script files that cannot be executed will NOT be marked and hence you will be awarded 0 marks. You may wish to comment out the parts in your code which cause execution errors.
7. Include your name as author in the comments of all your submitted source files. For example, include the following block of comments at the beginning of each source file you need to submit.

```
# Name    : BAI She Jing
# Email ID: shejing.bai.2020
```

Instructions on how to submit your solutions:

1. When the test ends, follow the instructions given by your invigilator to zip up all the files required for submission in a zip archive. The name of the zip archive should be your email ID. For example, if your email is `shejing.bai.2020@sis.smu.edu.sg`, you should name the archive as **`shejing.bai.2020.zip`**. You may be penalized for not following the instructions given to you.
2. Once everybody has zipped up his/her files, your invigilator will instruct you to submit your solutions as a single zip file to eLearn under Assignments.

Question 1 (Difficulty Level: ★) [7 marks]

Implement a function called `get_all_tank_sizes`. This function takes in 1 parameter:

- `tanks` (type: `list`): A list of tuples. Each tuple represents the dimension (in inches) of a fish tank and is of the form `(length, width, height)`, where `length`, `width` and `height` are either of type `int` or of type `float`.

The absolute volume of a fish tank (in gallons) is calculated using the following formula. (One gallon is equal to 231 cubic inches.)

$$\text{volume} = \frac{\text{length} \times \text{width} \times \text{height}}{231}$$

The absolute volume will then be rounded down to an integer. For example, 19.8 is rounded down to 19 and 5.0 is simply rounded to 5.

A fish tank is categorized into one of three sizes (small, medium, and large) based on the following criteria:

a. If the volume (after rounding down) is less than 20 gallons, this is a **small** size fish tank.
b. If the volume (after rounding down) is between 20 (inclusive) and 50 (exclusive) gallons, this is a **medium** size fish tank.
c. If the volume (after rounding down) is 50 gallons (inclusive) or more, this is a **large** size fish tank.

The function returns the sizes (`'S'` for small, `'M'` for medium, `'L'` for large) of the tanks in the same order as their dimensions are given in `tanks`.

You can assume that the values for length, width and height are always positive values.

If the given list `tanks` is empty, this function returns an empty list.

Example 1:
If the function is invoked like this:

```
print(get_all_tank_sizes([(12, 6, 8),(24, 12, 18),(36.375, 18.375, 19)]))
```

the statement generates the following output:

```
['S', 'M', 'L']
```

Question 2 (Difficulty Level: ★) [6 marks]

Implement a function called `get_hi_lo`. This function takes in 1 parameter:

- `products` (type: `list`): a list of products. Each product is a tuple of the form `(name, price_in_cents)`, where `name` is of type `str` and `price_in_cents` is of type `int`. Note: You can assume that there are no two products of the same price.

The function returns a tuple `(a, b)` where `a` (type: `str`) is the name of the product with the highest price, and `b` (type: `str`) is the name of the product with the lowest price.

If the parameter `products` is empty, this function returns the tuple `(None, None)`.

Example 1:

If the function is invoked like this:

```
print(get_hi_lo([('apple', 100), ('orange', 200), ('pear', 300)]))
```

the statement generates the following output:

```
('pear', 'apple')
```

Note:

- `'pear'` has the highest price (300).
- `'apple'` has the lowest price (100).

Example 2:

If the function is invoked like this:

```
print(get_hi_lo([('apple',100),('orange',250),('pear',200),('banana',150)]))
```

the statement generates the following output:

```
('orange', 'apple')
```

Example 3:

If the function is invoked like this:

```
print(get_hi_lo([('apple', 100)])
```

the statement generates the following output:

```
('apple', 'apple')
```

Note:

- `'apple'` has both the highest price and the lowest price.

Question 3 (Difficulty Level: ★★) [3 marks]

Implement a function called `calculate_term_gpa`. This function takes 2 parameters:

- `term_grades` (type: `str`). This string contains the grades of a student for this semester. Each grade is an uppercase letter (`'A'` to `'Z'`) optionally followed by a non-letter character (e.g., `'*'`, `'+'`, `'1'`) depending on the grading system. In other words, each grade is either a single uppercase letter, or an uppercase letter followed by a non-letter character. `term_grades` consists of a sequence of such grades. E.g., if `term_grades` is `'AB+C-'`, it means the student took three courses this semester and got A, B+ and C- for these three courses.
- `mapping` (type: `dict`). In this dictionary, the keys are the possible grades (e.g., `'A'`, `'B+'`) and the values are the grade points corresponding to those grades (e.g., 4.0 for `'A'`, 3.3 for `'B+'`). The grades and their corresponding grade points stored in this dictionary depend on the grading system.

You can assume that `mapping` contains all grades that appear in `term_grades`.

The function returns the term GPA of this student. The term GPA is the average grade point over all the courses the student has taken this semester. It can be computed using the following formula:

$$\frac{\text{GradePoint}_1 + \text{GradePoint}_2 + \ldots + \text{GradePoint}_n}{n}$$

where n is the number of courses taken by the student.

Note:

- You can assume that `term_grades` is NOT an empty string, i.e., `term_grades` contains at least one grade.
- You will receive some partial marks if your function can handle `term_grades` consisting of only single-letter grades, e.g., `'ACAC'`.

Example 1:

If the function is invoked like this:

```
print(calculate_term_gpa('ACAC', {'A':4, 'B':3, 'C':2, 'F':0}))
```

the statement generates the following output:

```
3.0
```

Note:

- The student took 4 courses and scored two A's and two C's.
- GPA is (4 + 2 + 4 + 2)/4 = 3.0

Example 2:

If the function is invoked like this:

```
print(calculate_term_gpa('A+AA-',
    {'A+':4.3, 'A':4.0, 'A-':3.7, 'B+':3.3, 'B':3.0, 'B-':2.7,
     'C+':2.3, 'C':2.0, 'C-':1.7, 'D+':1.3, 'D':1.0, 'F':0}))
```

the statement generates the following output:

```
4.0
```

Note:

- The student took 3 courses and scored one A+, one A and one A-.
- GPA is (4.3 + 4 + 3.7)/3 = 4.0

Example 3:

If the function is invoked like this:

```
print(calculate_term_gpa('A*AA',
    {'A*':4, 'A':3.5, 'B*':3, 'B':2.5, 'C':2, 'F':0}))
```

the statement generates the following output:

```
3.6666666666666665
```

Note:

- The student took 3 courses and scored one A* and two A's.
- GPA is (4 + 3.5 + 3.5)/3 = 3.6666666666666665

Question 4 (Difficulty Level: ★ ★) [3 marks]

Implement a function called `get_daily_spending`. This function takes in 5 parameters:

- `filename` (type: `str`): This is the name of a file containing transaction data. The format of the file will be explained below.
- `start_day` (type: `int`): A start day, which is an integer between 1 (inclusive) and 31 (inclusive), e.g., 5 or 31.
- `end_day` (type: `int`): An end day, which is also an integer between 1 (inclusive) and 31 (inclusive), e.g., 1 or 13. You can assume that the `end_day` is always larger than or equal to the `start_day`.
- `month` (type: `int`): A month of the year, which is an integer between 1 (inclusive) and 12 (inclusive), e.g., 5 or 12.
- `year` (type: `int`): A year, represented by a 4-digit integer, e.g., 2020.

You can assume that all parameters `filename`, `start_day`, `end_day`, `month` and `year` are valid.

The function will return a list that contains the daily spending (in cents) from `start_day` (inclusive) to `end_day` (inclusive) of the specified month and year. For example, if `start_day` is 3 and `end_day` is 5, the returned list is to contain the daily spending (in cents) of day 3, 4 and 5 of the specified month and year.

The `filename` refers to a text file that contains transaction data. Each transaction is in one row with three columns separated by the pipe character (`'|'`).

- The first column is the date of the transaction in `'dd/mm/yyyy'` format. However, month and day can be either 1 digit or 2 digits. E.g., 8 May 2020 can be represented by `'8/5/2020'`, `'08/5/2020'`, `'8/05/2020'` or `'08/05/2020'`.
- The second column is a whole number representing the transaction amount (in cents).
- The last column is a description of the transaction.

An example of the file is as follows:

```
04/05/2020|100|apple
3/10/2020|3450|phone bill
4/5/2020|150|orange
03/10/2020|8050|birthday gift
4/10/2020|1520|meal
04/10/2020|30000|holiday air ticket
5/10/2020|100|pear
3/10/2020|2000|EZ-link card
```

Example 1:

If the function is invoked like this where `'trans_file.txt'` contains the transactions above

```
print(get_daily_spending('trans_file.txt', 3, 5, 10, 2020))
```

the statement generates the following output:

```
[13500, 31520, 100]
```

Note:

- The spending on 3/10/2020 is 13500 cents (from the 2nd, 4th and last rows of the file).
- The spending on 4/10/2020 is 31520 cents (from the 5th and 6th rows of the file).
- The spending on 5/10/2020 is 100 cents (from the 7th row of the file).

Example 2:

If the function is invoked like this:

```
print(get_daily_spending('trans_file.txt', 6, 7, 10, 2020))
```

the statement generates the following output:

```
[0, 0]
```

Note:

- There is no spending record on 6/10/2020 and 7/10/2020.

Question 5 (Difficulty Level: ★★★) [6 marks]

Part A

Implement a function called `multiply()`. This function takes 1 parameter:

- `polynomials` (type: `list`): This is a list of lists, where each list inside `polynomials` represents the coefficients of a single-variable polynomial such as $-2x^2 + 3x + 1$. (This means you will NOT see polynomials with more than one variable such as $2xy + 3x + 5$.)

  Each polynomial is represented as a list of the form [a_n, a_{n-1}, a_{n-2}, …, a_2, a_1, a_0], where a_n is the coefficient of x^n, a_{n-1} is the coefficient x^{n-1}, and so on. You can assume that the list consists of all the coefficients of the polynomial (i.e., coefficients of all individual terms including the zero value coefficients as shown in the third illustration below). You can also assume that all coefficients are integers.

  For example,
  - `[2, 6]` represents the polynomial $2x + 6$.
  - `[-2, 3, 1]` represents the polynomial $-2x^2 + 3x + 1$.
  - `[5, 0, 3, 7]` represents the polynomial $5x^3 + 3x + 7$.

  Note:
  - The list representing a polynomial will NOT be empty.
  - The first element of the list will not be 0 (e.g., $2x + 6$ will not be represented as `[0, 2, 6]`), **except** for the special case when we represent a zero polynomial; in the case of a zero polynomial, we use `[0]` (a list with a single element 0 inside) to represent it.

  The parameter `polynomials` is a list of polynomials. E.g., if `polynomials` is `[ [2, 6] , [-2, 3, 1], [5, 0, 3, 7] ]`, it represents the following list of polynomials: [$2x + 6$, $-2x^2 + 3x + 1$, $5x^3 + 3x + 7$] .

This function returns a list that represents coefficients of the resultant polynomial after **multiplying** all the polynomials in the given list `polynomials`.

To multiply 2 polynomials, multiply each term in the first polynomial by each term in the second polynomial. For example,

$$
\begin{aligned}
&(\mathrm{x}^2 + 2\mathrm{x} + 3)(5x^2 + 6x + 7) \\
&= \mathrm{x}^2(5\mathrm{x}^2 + 6\mathrm{x} + 7) \quad + 2\mathrm{x}(5\mathrm{x}^2 + 6\mathrm{x} + 7) \qquad + 3(5\mathrm{x}^2 + 6\mathrm{x} + 7) \\
&= (5x^4 + 6x^3 + 7x^2) + (10x^3 + 12x^2 + 14x) + (15x^2 + 18x + 21) \\
&= 5\mathrm{x}^4 + 16\mathrm{x}^3 + 34\mathrm{x}^2 + 32\mathrm{x} + 21
\end{aligned}
$$

Note:
- You can assume that the list `polynomials` always contains at least one element.
- If the result is a zero polynomial, the function should return `[0]` (rather than `[0, 0]`, `[0, 0, 0]`, etc.).

Example 1:

If the function is invoked like this:

```
polynomials = [[1, 2, 3], [5, 6, 7]]
print(multiply(polynomials))
```

the statement generates the following output:

```
[5, 16, 34, 32, 21]
```

Example 2:

If the function is invoked like this:

```
polynomials = [[1, 2], [3, 4, 5], [6, 7, 8]]
print(multiply(polynomials))
```

the statement generates the following output:

```
[18, 81, 172, 231, 174, 80]
```

Note: We get the output above because
$(x + 2) \cdot (3x^2 + 4x + 5) \cdot (6x^2 + 7x + 8) = 18x^5 + 81x^4 + 172x^3 + 231x^2 + 174x + 80$

Example 3:

If the function is invoked like this:

```
polynomials = [[1, 2], [0], [3, 4, 5], [6, 7, 8]]
print(multiply(polynomials))
```

the statement generates the following output:

```
[0]
```

Note: We get the output above because
$(x + 2) \cdot 0 \cdot (3x^2 + 4x + 5) \cdot (6x^2 + 7x + 8) = 0$

Part B

Implement a function called `get_polynomial()`. This function takes 1 parameter:

- `poly_str` (type: `str`): This is a string representing an algebraic expression that is a polynomial *with a single variable*. For example, the string `'2x - 2x^2 + 3x - 1 + 6x^3'` represents $2x - 2x^2 + 3x - 1 + 6x^3$.
  - You can assume that `poly_str` contains only the following characters: digits (`'0'` to `'9'`), `'x'`, `'^'`, `'+'`, `'-'`, and spaces.
  - You can assume that there is always either a single `'+'` or a single `'-'` between two terms in the expression.
  - Each term in the expression has a coefficient, the variable `x` (except for those terms that represent constants such as `4` and `-9`), and an exponent (except for those terms that represent constants and those terms where the exponent is 1, such as `4x` and `-9x`). For example, `-2x^3` has a coefficient value of `-2` and an exponent value of `3`.
  - You can assume that a coefficient in the given `poly_str` is never `0`. E.g., you will not have something like `'2x - 0x^2'`.
  - You can assume that an exponent is always a positive integer.
  - There may be either no space or an arbitrary number of spaces before/after each `'+'` or `'-'`.
  - There is no space between a coefficient and its corresponding `'x'`, between `'x'` and `'^'`, or between `'^'` and its corresponding exponent.

This function returns a list that represents all the coefficients of the polynomial **after it is reduced to the simplest form**. In other words, the function returns a list [a_n, a_{n-1}, a_{n-2}, …, a_2, a_1, a_0], where a_n is the coefficient of x^n, a_{n-1} is the coefficient x^{n-1}, and so on. If the polynomial's degree (after simplification) is n, then the returned list must have $n + 1$ elements, corresponding to the coefficients of $x^n, x^{n-1}, \dots, x$, and the constant.

For example, if `poly_str` is `'2x - 2x^2 + 3x - 1 + 6x^3'`, then this function will return `[6, -2, 5, -1]`. If `poly_str` is `'3x^2 - 2x - 3x^2'`, then the function will return `[-2, 0]` instead of `[0, -2, 0]`. This is because after simplifying the expression, we no longer have any x^2 term. (You may still receive some partial marks if your function returns `[0, -2, 0]`.)

Example 1:

If the function is invoked like this:

```
poly_str = '2x - 2x^2 + 3x - 1 + 6x^3'
print(get_polynomial(poly_str))
```

the statement generates the following output:

```
[6, -2, 5, -1]
```

Example 2:

If the function is invoked like this:

```
poly_str = '  4x -2x^2 +   3x^5 -   6x^2'
print(get_polynomial(poly_str))
```

the statement generates the following output:

```
[3, 0, 0, -8, 4, 0]
```