Programming I (Python)
Assignment 5

Instructions

- Please use Python 3 for all your implementations.
- Answers to each question should be provided in a file whose name is mentioned against the respective question.
- This assignment is about functions. **Please ensure that your code does not have any extraneous input/output code.**
- In several questions, underscores (`'_'`) have been used to highlight spaces (`' '`) in the output code. Your output should contain the space character (`' '`) in all those spaces.
- **How to submit.**
  1. The stub/starter files for all questions (except Q. 1) are provided in the directory named `answers`. Please write your answers in the files with appropriate names as given in the questions.
  2. Once you are satisfied with your solutions/answers, exit the `answers` directory and compress the `answers` directory preferably using the following command:

     ```
     tar cvzf answers.tar.gz answers
     ```

  3. Upload `answers.tar.gz` as the submission to the assignment.

Theory Questions

1. In the following pieces of code, please identify if they have side-effect or not. If yes, please mention/mark the precise point where we see side-effect.

   (a)
   ```
   def add(x, y): return x + y
   ```

   (b)
   ```
   x = int(input("Enter the number:" ))
   ```

(c)
```
print("Hello world!")
```

(d)
```
sum = 0
for i in range(10):
  sum += i
print(sum)
```

(**file:** `Q1.pdf` / `Q1.doc`/ `Q1.docx`)

Multi Procedure Programs

2. Your functions are expected to function only with positive number. Respond to negative numbers may be left undefined.

   Implement the functions `increment` and `decrement` with the usual meanings.
   For example:

```
print(increment(1))
print(decrement(1))
```

   will print

```
2
0
```

   Solution:

   (a)
   1. Correct answer for `increment` - 1 mark
   2. Correct answer for `decrement` - 1 mark

(b) Implement the functions `add` and `subtract` using `increment` and `decrement` respectively.
   For example:

```
print(add(1, 2))
print(subtract(1, 2))
```

   will print

```
3
-1
```

   Solution:

   1. Correct answer for `add` - 1 mark
   2. Correct answer for `subtract` - 1 mark
   3. `add` calls `increment`. - 2 marks
   4. `subtract` calls `decrement`. - 2 marks

(c) Implement the functions `multiply` and `divide` using `add` and `subtract` respectively.
For example:

```
print(multiply(5, 6))
print(divide(7, 2))
```

will print

```
30
3
```

Solution:

1. Correct answer for `multiply` - 1 mark
2. Correct answer for `divide` - 1 mark
3. `multiply` calls `add`. - 2 mark
4. `divide` calls `subtract`. - 2 mark

(d) Implement the function `exponent` using `multiply`.
For example:

```
print(exponent(2, 3))
```

will print

```
8
```

Solution:

1. Correct answer for `exponent` - 1 mark
2. `exponent` calls `multiple`. - 2 marks

(**file:** `Q2.py`)

3. Consider mathematical expressions in the form of sum-of-products (SOP). For example, $(1 \times 2 \times 3) + (20 \times 40)$ is a SOP expression. We represent a SOP expression as a list of lists, where each inner list represents one product term. For example, $(1 \times 2 \times 3) + (20 \times 40)$ is represented as `[[1, 2, 3], [20, 40]]`. The goal of this question is to come up with a function `evaluate_SOP` that, given an expression e, finds its numerical value. We approach the solution to the problem in stepwise manner by designing functions that solves various parts of this problem.

(a) Write a function `product_of_list` that computes the product of all numbers in a list. For example: Example:

```
print(product_of_list([1, 2, 3]))
```

will print

```
6
```

Solution:

1. Correct answer for `product_of_list` - 2 mark

(b) Write a function `reduce_terms` that takes an expression e reduces each product term into its value. `reduce_terms` should use `product_of_list` to compute the value of each term. For example: Example:

```
print(reduce_terms([[1, 2, 3], [20, 40]]))
```

will print

```
[6, 800]
```

Solution:

1. Correct answer for `reduce_terms` - 1 mark
2. `reduce_terms` calls `product_of_list`. - 2 marks

(c) Write a function `sum_of_list` that computes the sum of all numbers in a list. For example: Example:

```
print(sum_of_list([6, 800]))
```

will print

```
806
```

Solution:

1. Correct answer for `sum_of_list` - 1 mark

(d) Write a function `evaluate_SOP` that computes the value of an expression e provided in a SOP form. For example:

```
print(evaluate_SOP([[1, 2, 3], [20, 40]]))
```

will print

```
806
```

`evaluate_SOP` should implement the architecture shown in Fig. 1.

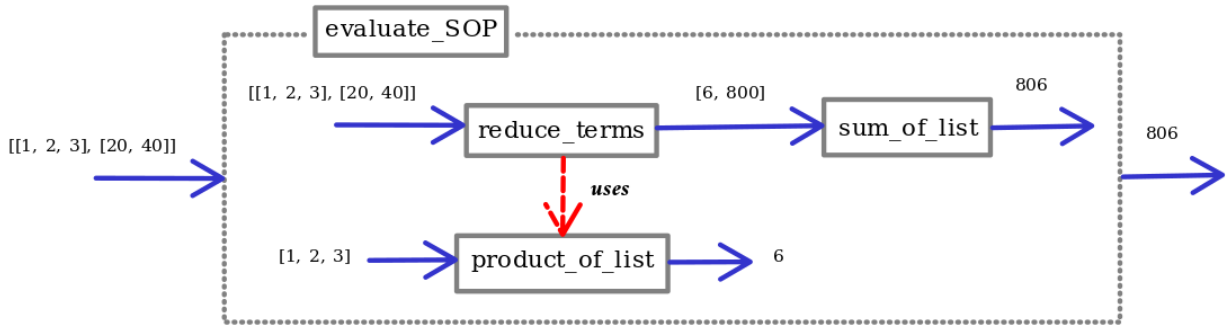


Figure 1: **Architecture of** `evaluate_SOP`: `evaluate_SOP` calls `reduce_terms` which achieves its functionality by in turn calling `product_of_list`
. `evaluate_SOP` gives the output obtained from `reduce_terms` to `sum_of_list` and directly returns the result returned therefrom as its own result.

Solution:

1. Correct answer for `evaluate_SOP` - 2 mark
2. `evaluate_SOP` calls `sum_of_list`. - 2 marks
3. `evaluate_SOP` calls `reduce_terms`. - 2 marks

(**file:** `Q3.py`)

4. **Note:** For each function, its implementation should adhere to the function call graph indicated against it. In a function call graph, functions are shown by rectangular boxes and a procedure `f` calling another procedure `g` is shown as follows:

(a) `is_wellformed`: Takes a list of lists and checks if it is a well formed matrix. The condition for well-formedness is that all rows should be of the same length. Returns `True` if found to satisfy the above conditions; returns `False` otherwise.

For example:

```
print(is_wellformed([[1, 2, 3], [20, 40, 50]]))
print(is_wellformed([[1, 2, 3], [20, 40]]))
```

will print

```
True
False
```

Solution:

1. Total marks - 3 mark

2. One test case for each branch of reference solution.

(b) `are_addable`: Takes two matrices and checks if they can be added. For this, it checks:

1. if both matrices are well formed using `is_wellformed`;
2. if both matrices have the same number of rows.
3. if both matrices have rows of equal length.

Returns `True` if found to satisfy the above conditions; returns `False` otherwise.

For example:

```
print(are_addable([[1, 2, 3], [20, 40, 50]], [[1, 2, 3], [20, 40, 50]]))
print(are_addable([[1, 2], [20, 40]], [[1, 2, 3], [20, 40, 50]]))
```

will print

```
True
False
```

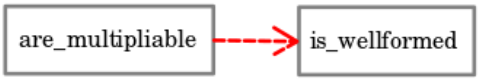
Solution:

1. Total marks - 3 mark
2. Call to `is_wellformed` - 2 mark
3. One test case for each branch of reference solution.

(c) `are_multipliable` Takes two matrices m_1 and m_2 and checks if they can be multiplied. For this, it checks:

1. if both m_1 and m_2 are well formed using `is_wellformed`;
2. if the number of columns in m_1 is equal to the number of rows in m_2.

Returns `True` if found to satisfy the above conditions; returns `False` otherwise.



For example:

```
print(are_multipliable([[1, 2, 3], [20, 40, 50]], [[1, 2, 3], [20, 40, 50]]))
print(are_multipliable([[1, 2], [20, 40]], [[1, 2, 3], [20, 40, 50]]))
```

will print

```
False
True
```

Solution:

1. Total marks - 3 mark
2. Call to `is_wellformed` - 2 mark
3. One test case for each branch of reference solution.

(d) `scalar_multiply_list`: Takes an integer n and a list l as input parameters, and returns a new list l' such that length of l is equal to length of l', and each element of l' is n times the corresponding element of l.

For example:

```
print(scalar_multiply_list(3, [1, 2, 3]))
```

will print

```
[3, 6, 9]
```

Solution:

1. Correct output `scalar_multiply_list` - 2 mark

(e) `scalar_multiply_matrix`: Takes an integer n and a matrix m, and returns a new matrix m' such that the dimensions of m are equal to the dimensions of m', and each element of m' is n times the corresponding element of m. `scalar_multiply_matrix` must use `scalar_multiply_list` to process each of its rows.

For example:

```
print(scalar_multiply_matrix(3, [[1, 2, 3]]))
```

will print

```
[[3, 6, 9]]
```

Solution:

1. Correct output `scalar_multiply_matrix` - 2 mark
2. Call to `scalar_multiply_list` - 2 mark

(f) `add_lists`: Takes two lists l_1 and l_2 of equal lengths as input paramaters, and returns a new list l such that $|l| = |l_1| = |l_2|$ and $\forall i \; s.t. \; 0 \leq i < |l|, l[i] = l_1[i] + l_2[i]$.

For example:

```
print(add_lists([1, 2, 3], [4, 5, 6]))
```

will print

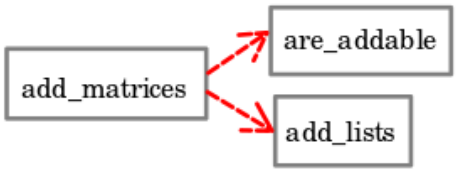
```
[5, 7, 9]
```

Solution:

1. Correct output `scalar_add_lists` - 2 mark

(g) `add_matrices`: Takes two lists m_1 and m_2 such that $||m_1|| = ||m_2||$[1] as input paramaters, and returns a new matrix m such that $||m|| = ||m_1|| = ||m_2||$ and $\forall i \ s.t. \ 0 \leq i < \#rows(m), j \ s.t. \ 0 \leq j < \#columns(m), \ m[i][j] = m_1[i][j] + m_2[i][j]$.

1. `add_matrices` must use `are_addable` to check their addability before proceeding to add m_1 and m_2. It should proceed only if they are found to be addable; otherwise, it should print an appropriate error message and exit.
2. `add_matrices` must use `add_lists` to add each individual row.



For example:

```
print(add_matrices([[1, 2, 3]], [[4, 5, 6]]))
```

will print

```
[[5, 7, 9]]
```

Solution:

1. Correct output `scalar_add_matrices` - 2 mark
2. Call to `scalar_are_addable` - 2 mark
3. Call to `add_lists` - 2 mark

(h) `multiply_lists`: Takes two lists l_1 and l_2 of equal lengths as input paramaters, and returns a new list l such that $|l| = |l_1| = |l_2|$ and $\forall i \ s.t. \ 0 \leq i < |l|, l[i] = l_1[i] * l_2[i]$.

For example:

```
print(multiply_lists([1, 2, 3], [4, 5, 6]))
```

will print

[1] $||m||$ represents the dimensions of a matrix m.

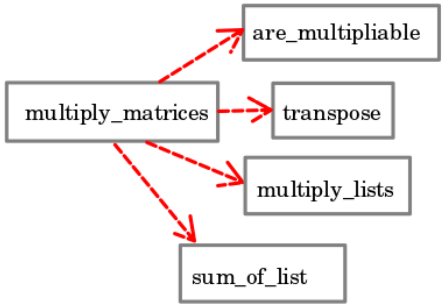
```
[4, 10, 18]
```

Solution:

1. Correct output `multiply_lists` - 2 mark

(i) `multiply_matrices`: Takes two matrices m_1 and m_2 as input parameters and returns their product.

1. `multiply_matrices` must use `are_multipliable` to check their addability before proceeding to add them. It should proceed only if they are found to be multipliable; otherwise, it should print an appropriate error message and exit.
2. `multiply_matrices` must use `transpose` to obtain m_2^T by transposing m_2.
3. `multiply_matrices` must use `multiply_lists` to multiply each individual row of m_1 and m_2^T.



For example:

```
print(multiply_matrices([[1, 2, 3],[4, 5, 6]],
 [[7, 10], [8, 11], [9, 12]])))
```

will print

```
[[50, 68], [122, 167]]
```

Solution:

1. Correct output `multiply_matrices` - 3 mark
2. Call to `transpose` - 2 mark
3. Call to `are_multipliable` - 2 mark
4. Call to `multiply_lists` - 2 mark
5. Call to `sum_of_list` - 2 mark

(**file:** `Q4.py`)