CS1010X — Programming Methodology
National University of Singapore

Practical Examination

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**. Note that the various methods of sequences (tuple, set, list, dict) that you may need are listed in the first few pages of our **Lecture 8**.
2. You are to do your work without any assistance from another intelligent human being – if found otherwise, you will receive **ZERO** for the practical exam and will be subject to other disciplinary actions.
3. This practical exam consists of **4** "topics" printed in **11** pages (inclusive of this cover page).
4. The maximum score of this quiz is **32 marks** and you will need to answer all questions to achieve the maximum score. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question. You are advised to use your time wisely and do move on to other parts if one part is difficult and can take up lots of time to solve.
5. Your answers should be submitted on Coursemology.org **BEFORE** the end of the exam. If you have any submissions timestamped with a time after the exam has ended, your submission for that question will not be graded. Remember to **finalize** your submissions before the end of the exam.
6. You will be allowed to run some public tests cases to verify that your submission is correct. Note that you can run the test cases on Coursemology.org up to a **maximum of 5 times** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness using IDLE and not on Coursemology.org. Do however ensure that you submit your answers correctly by running the test cases at least once.
7. You are also provided with the template `practical-template.py` to work with. If Coursemology.org fails, you need to rename your file `practical-<mat_no>.py` where `<mat_no>` is your matriculation number and submit that file by leaving it in your computer.
8. Please note that while sample executions are given in some cases, it is not sufficient to write programs that simply satisfy the given examples. Your programs will be tested on other inputs and they should exhibit the required behaviours as specified by the problems to get the allocated credit.
9. Please behave like a good programmer to make your codes readable with good naming convention for the variables and function names. If you do not do so, we reserve the right to deduct small credit even if your program is correct.

GOOD LUCK!

Topic 1: Secret Message [9 marks]

Violet works as an Auto Memory Doll – a ghostwriter for people who cannot write or are looking for help expressing their emotions in letters. Now she wants to write a secret message that can't be read by anyone except the recipient.

Violet studied many encryption methods: Caesar cipher, Chaocipher, Hill Cipher... Finally she decided to use such a method:

The original idea of the transformation is simply aligning two alphabets: the plain alphabet and the cipher alphabet. However, the message encrypted with such a naïve method may be decrypted by others easily. Violet uses a dynamic method instead. Each time she encrypted a letter from plain to cipher, she moves the letter to the front of the plain alphabet. For instance, if the plain and cipher alphabets are like:

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

and Violet is going to encrypt the word `HELLO`, she will encrypt the letter `H` first and get the cipher `K` and move `H` to the front of the plain alphabet (while there is no change to the cipher alphabet). Now the alphabets will become:

Plain	H	A	B	C	D	E	F	G	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

and the next letter `E` will be encrypt to `I`. This process repeats until all the letters are encrypted. In the end, `HELLO` will be encrypted to `KIODR` (where the first `L` in `HELLO` is encrypted as `O` while the second `L` is `D` due to the move to the front after the first `L` is encrypted).

Question 1. Please write a function `encrypt(message,cipher)` to help Violet encrypt a message with the cipher alphabet. The initial plain alphabet is in alphabetical order and assume that there are only capital letters and spaces in the message. Note that each space still remains a space in the encrypted message. [3 marks]

Question 2. Given the secret message from Violet and the cipher alphabet, can you write a function `decrypt` to help the recipient decrypt the secret message? [3 marks]

Violet realized that encrypting the letters one by one is quite troublesome – she can do this word by word plus the similar trick of moving to front after encrypting each word! For instance, if she is going to encrypt `HELLO MAJOR GILBERT` and the initial alphabets are:

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

she can encrypt `HELLO` to `KHOOR` first, then move the appeared letters (`E`, `H`, `L` and `O`) to the front of the plain alphabet in alphabetical order. The alphabets will become:

Plain	E	H	L	O	A	B	C	D	F	G	I	J	K	M	N	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

and the next word `MAJOR` will be encrypted to `QHOGU`. After that, Violet will move the appeared letters in this word (`A`, `J`, `M`, `O` and `R`) to the front of the plain alphabet again and the alphabets will become:

Plain	A	J	M	O	R	E	H	L	B	C	D	F	G	I	K	N	P	Q	S	T	U	V	W	X	Y	Z
Cipher	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Then, the last word `GILBERT` will be encrypted to `PQKLIHW` (and the alphabets are updated accordingly). In summary, the original message `HELLO MAJOR GILBERT` now will be encrypted to `KHOOR QHOGU PQKLIHW`.

Violet learned how to use modern technology (Python) to do the encryption from you (and we shall skip this encryption function). However, it might be difficult for the recipient to decrypt the secret message and you are asked to help here.

Question 3. Given the cipher alphabet and the secret message encrypted word by word from Violet, can you write a function `decrypt_wbw` to help the recipient decrypt the secret message? To split a secret message into a list of words, you may use the `string.split()` method. E.g.

```
>>> string = "a bc def ghij 1 23 456"
>>> string.split()
['a', 'bc', 'def', 'ghij', '1', '23', '456']
```

[3 marks]

Hope the above exercises do warm up the place now. ”Emm...all these parts are 1D exercises,” Violet talking to herself, ”so it is time to move on to 2D and then higher dimension !”

Topic 2: Rune Inventor [6 marks]

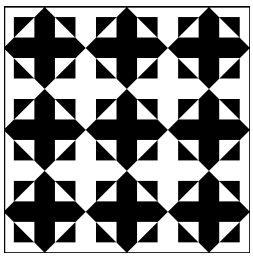
Violet met a rune inventor when she was traveling around the continent. She has seen many beautiful runes and would like to create her own rune. She asked the rune inventor for advice and the rune inventor taught her some elementary skills.

"You know what? The world is digital," the inventor said, "so if you can control the grids, you can do everything." Here is the program the inventor gave Violet, which creates a rune of an n-by-n pattern of pictures:

```
from runes import *
def stackn(n,pic):
    if n == 1:
        return pic
    else:
        return stack_frac(1/n, pic, stackn(n-1, pic))

def nxn(n,pic):
    return stackn(n,quarter_turn_right(stackn(n,quarter_turn_left(pic))))
```

For example, the result of `show(nxn(3, make_cross(rcross_bb)))` is as follows:



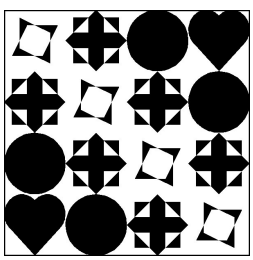
Notice that the result is done through two parts. First, it has the function `stackn` to create a column of crosses, and then use it in the function `nxn` to create the result.

"Emm... This is quite easy..." Violet thought, "I can do something more advanced." She wants to create a rune matrix where each element is from a list of pictures, i.e. she wants to create a new function `mxn_matrix` which accepts a list of pictures and a matrix of indices, called *index matrix*, where each index in the matrix is the index of the corresponding picture in the list of pictures.

For example, the result of the expression

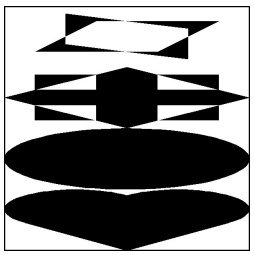
```
show(mxn_matrix([make_cross(nova_bb), make_cross(rcross_bb), circle_bb, heart_bb],\
        [[0, 1, 2, 3], [1, 0, 1, 2], [2, 1, 0, 1], [3, 2, 1, 0]]))
```

is as follows:



Note that the index matrix is a list of lists where the first element (`[0, 1, 2, 3]` in our example) of the list is for the first row, and the second element (`[1, 0, 1, 2]` in our example) is for the second row, and so on.

Following what Violet has learned from the inventor, she first needs a new function `stackn_list` which accepts a list of pictures and creates a column of pictures, where the pictures appear from top to bottom in the same order as in the input list. For example, `stackn_list([make_cross(nova_bb), make_cross(rcross_bb), circle_bb, heart_bb])` is:



Then, if Violet writes the following expressions:

```
show(stackn_list([make_cross(rcross_bb), make_cross(nova_bb), pentagram_bb,\
            make_cross(nova_bb), make_cross(rcross_bb)]))

show(mxn_matrix([make_cross(rcross_bb), make_cross(nova_bb), pentagram_bb],\
                [[0, 0, 0, 0, 0, 0], \
                 [0, 1, 1, 1, 1, 0], \
                 [0, 1, 2, 2, 1, 0], \
                 [0, 1, 1, 1, 1, 0], \
                 [0, 0, 0, 0, 0, 0]] ))
```

she will get a column of 5 pictures for the first expression and a 5×6 rune for the second expression:

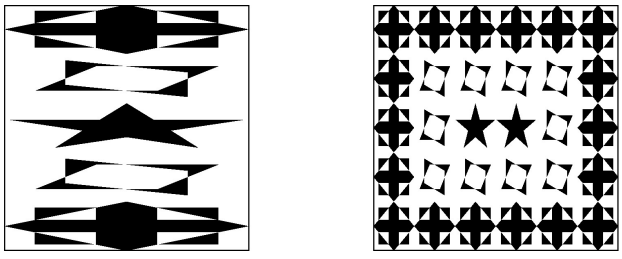


Figure 1: Left is the result of `stackn_list` and right is `mxn_matrix`

Question 4. Please implement the `stackn_list` function which accepts a list of pictures and returns a column of pictures. [3 marks]

Question 5. Please implement the `mxn_matrix` function which accepts a list of pictures and an index matrix and returns the corresponding rune in accordance to the index matrix. [3 marks]

Topic 3: Multidimensional Sum [9 marks]

When Violet was learning Python, she found an interesting problem which is to compute the sum of the rows or columns of a matrix. Specifically, if she wants to compute the sum of the elements in each row of a matrix, she can implement a `row_sum` function which takes in a matrix and returns a list, where the i-th element is the sum of the elements in the i-th row of the matrix:

```
def row_sum(matrix):
    return list(map(lambda row: sum(row), matrix))
```

Similarly, if she wants to compute the sum of the elements in each column of a matrix, she can implement a `col_sum` function as follows:

```
def col_sum(matrix):
    if matrix == []:
        return []
    res = []
    for i in range(len(matrix[0])):
        res.append(sum(map(lambda row: row[i], matrix)))
    return res
```

"OK, this seems like an easy task for this case of a 2D matrix which we can use 2 separate functions, one for row and one for column." Violet thought, "Then what if we have a general list with arbitrary number of dimensions and not just a list which is 1D nor a list of lists which is only 2D? Can I implement a single function to compute the sum along any axis of an arbitrary dimensional array?"

Her goal at the end of this quest is to implement `sum_along(axis,arr)` where `axis` indicates the dimension to do the sum, and `arr` is the input arbitrary dimensional array. Note that, for a 2D array, the `sum_along(0,arr)` should have the same output as `col_sum(arr)`, and `sum_along(1,arr)` has the same output as `row_sum(arr)`. Let's help to do this in a step-by-step manner.

We first need to understand the generalization of matrix to *array* in case you don't know. In 1D, an array `arr` is simply a list of numbers in Python. So, the `sum_along` is simply summing (along `axis=0`) all the numbers (which are referred to as `arr[i]`) in the input list `arr`. In 2D, the array `arr` is the mentioned $m \times n$ matrix in the `row_sum` and `col_sum`, and the element of the array is referred to as `arr[i][j]` where `i` is the row and `j` is the column. So, the sum along the column is the sum along the `axis=0` with output of an 1D array of n elements, and the sum along the row is the sum along the `axis=1` with output of an 1D array of m elements.

From the above, we now can understand then a 3D array `arr` refers to a list of lists of lists. More importantly, an element in a 3D array is referred to with 3 indices as `arr[i][j][k]` and the sum along can be performed along `axis=0` w.r.t the first index (`i`), along `axis=1` w.r.t the second index (`j`), etc. Then, an element in a 4D array `arr` is referred as `arr[i][j][k][`ℓ`]`, and likewise for an element of an d-dimensional array is referred with d indices. To make it simple for our working (when we do things in recursion), we shall write `arr[i][j]` as `arr[i,j]`; `arr[i][j][k]` as `arr[i,j,k]`; and `arr[i][j][k][`ℓ`]` as `arr[i,j,k,` ℓ`]` and so on from here on.

We next need to understand the *shape* of an d-dimensional array. It is a list of d elements where its i-th element records the number of index values possible for the

i-th axis. Take for example, `lst2=[[1,2,3],[4,5,6]]`, it is a 2D array, and its shape is `[2,3]` because along `axis=0` there are 2 possible values (0 and 1) for the index, and along `axis=1` there are 3 values (0, 1 and 2). That is, we have the following indices to refer to elements of `lst2`: `[0,0]`, `[0,1]`, `[0,2]`, `[1,0]`, `[1,1]`, `[1,2]`. Note that we assume our d-dimensional array is proper in the sense that it contains `k` number of $(d-1)$-dimensional sub-arrays where each has the same shape as one another. So, the shape of this d-dimensional array can be obtained (in a recursive manner) as `[k]` concatenates with the shape of any one of its $(d-1)$-dimensional sub-arrays.

Question 6. Please implement the `get_shape(arr)` function which returns the shape of an array `arr`. It is guaranteed that the input array is non-empty. [1 marks]

Here are some sample executions where `lst1, lst1a` are 1D, `lst2` is 2D, `lst3` is 3D, and `lst4` is 4D. Note that all examples here are up to 4D but the `get_shape` function (and all subsequent ones) must work for higher dimensions too.

```
>>> lst1 = [1]
>>> get_shape(lst1)
[1]
>>> lst1a = [1, 2]
>>> get_shape(lst1a)
[2]
>>> lst2 = [[1, 2, 3], [4, 5, 6]]
>>> get_shape(lst2)
[2, 3]
>>> lst3 = [[[1, 2], [3, 4], [5, 6] ], [[7, 8], [9, 10], [11, 12]]]
>>> get_shape(lst3)
[2, 3, 2]
>>> lst4 = [ lst3 ]
>>> get_shape(lst4)
[1, 2, 3, 2]
```

Question 7. Please implement the `get_value(arr,idx)` function which returns the value of the element in the array `arr` with the given index `idx`. For `arr` an d-dimensional array, the index `idx` is a list of length d. We assume all inputs are good and do not need to handle any possible error. [1 marks]

```
>>> get_value(lst1, [0])
1
>>> get_value(lst1a, [1])
2
>>> get_value(lst2, [0,2])
3
>>> get_value(lst2, [1,1])
5
>>> get_value(lst3, [1,0,1])
8
>>> get_value(lst4, [0,1,0,1])
8
```

Question 8. Please implement the `set_value(arr,idx,val)` function which modifies the value of the element in the array `arr` with the given index `idx` to `val`. For `arr` an d-dimensional array, the index `idx` is a list of length d. We assume all inputs are good and do not need to handle any possible error. [1 marks]

```
>>> set_value(lst1, [0], 8)
>>> lst1
[8]
>>> set_value(lst1a, [1], 18)
>>> lst1a
[1, 18]
>>> set_value(lst2, [0,2], 28)
>>> lst2
[[1, 2, 28], [4, 5, 6]]
```

The above are some auxiliary functions Violet needs to solve the problem. We need 2 more as discussed next. For the simplest case of a 1D array, the output of `sum_along(axis, arr)` is simply a number equal to the sum of the elements in `arr`. In general, the output array is as follows:

- It has a shape of one less element than that of the shape of `arr`. Specifically, the shape of the output array is the same as that for the input `arr` but with the value of the selected `axis` removed. Continue with our example, the shape of `lst2` is `[2,3]`. Then `sum_along(0,lst2)` produces an output with the shape of `[3]`, and `sum_along(1,lst2)` of `[2]`. For the example of `lst4`, the `sum_along(1,lst4)` produces an output with the shape of `[1,3,2]`.
- The purpose of `sum_along(axis,arr)` is to reduce or remove the dimension of the `axis`. That is, it sums up values of elements along the selected `axis` to be a value for each distinct index without the selected `axis`. For `sum_along(1,lst3)`, each element of the output array (with shape `[2,2]`) is:

$$\texttt{output}[i,k] = \sum_{j=0}^{2} \texttt{lst3}[i,j,k] \quad \text{where } i \in \{0,1\} \text{ and } k \in \{0,1\}.$$

  Similarly, each element of the output array of `sum_along(1,lst4)` is:

$$\texttt{output}[i,k,\ell] = \sum_{j=0}^{1} \texttt{lst4}[i,j,k,\ell] \quad \text{where } i \in \{0\},\ k \in \{0,1,2\} \text{ and } \ell \in \{0,1\}.$$

Continuing from the above example, we have:

```
>>> lst3
[[[1, 2], [3, 4], [5, 6]], [[7, 8], [9, 10], [11, 12]]]
>>> sum_along(0, lst3)
[[8, 10], [12, 14], [16, 18]]
>>> sum_along(1, lst3)
[[9, 12], [27, 30]]
>>> lst4
[[[[1, 2], [3, 4], [5, 6]], [[7, 8], [9, 10], [11, 12]]]]
>>> sum_along(0, lst4)
[[[1, 2], [3, 4], [5, 6]], [[7, 8], [9, 10], [11, 12]]]
>>> sum_along(1, lst4)
[[[8, 10], [12, 14], [16, 18]]]
>>> get_shape( sum_along(1, lst4))
[1, 3, 2]
```

Question 9. Please implement the `create_arr(shape)` function which returns an array of the given `shape` with all its elements initialize to 0. [2 marks]

```
>>> get_shape(lst2))
[2, 3]
>>> lst2C = create_arr( get_shape(lst2) )
>>> lst2C
[[0, 0, 0], [0, 0, 0]]
>>> get_shape(lst2C)
[2, 3]
>>> get_shape(create_arr( get_shape(lst3)))
[2, 3, 2]
>>> create_arr( get_shape(lst3) )
[[[0, 0], [0, 0], [0, 0]], [[0, 0], [0, 0], [0, 0]]]
```

Question 10. Please implement the `next_idx(idx,shape)` function which returns the next available index after `idx` in accordance to the `shape` information. That is, this function is to return the next index in the "lexicographical order" that we normally know about numbers (e.g. 0000, 0001, 0002, ..., 0009, 0010, 0011, 0012, ..., 0019, 0020, ...) but limited by the possible values for each index in each dimension as given in `shape`. [2 marks]

```
>>> def listing_of_indices_given_a_shape (shape):
        idx = [0] * len(shape)
        print(idx)
        while idx != None:
            idx = next_idx(idx, shape) # you are to implement this function
            print(idx)

>>> listing_of_indices_given_a_shape([1,2,3,2])
[0, 0, 0, 0]
[0, 0, 0, 1]
[0, 0, 1, 0]
[0, 0, 1, 1]
[0, 0, 2, 0]
[0, 0, 2, 1]
[0, 1, 0, 0]
[0, 1, 0, 1]
[0, 1, 1, 0]
[0, 1, 1, 1]
[0, 1, 2, 0]
[0, 1, 2, 1]
None
```

With the above, Violet can put the parts together to solve the problem with a small Python program: Find the shape of the input array, create the output array (of shape with 1 less dimension) with all initial values equal to 0. Then go through each of the necessary index (using `next_idx`) to sum up the values of those required elements (of the input array) to be the value of the element of the output array with this index.

Question 11. Please implement the `sum_along(axis,arr)` function which returns a single value as discussed when `arr` is 1D, or returns an output array of 1 less dimension as `arr` with sums along the given `axis`. [1] [2 marks]

[1] Note that we will use our correct implementation of all the functions in earlier parts to test this part of your codes. That is, if any of your above is not correct, it will not affect your effort in this part.

Topic 4: The Matrix [8 marks]

Violet has learned how to represent a matrix (and higher dimensional data in the previous question) as a nested list, but she also realized that if the matrix is sparse (most of the elements in the matrix are zero), simply representing the matrix as a nested list will waste memory.

She wanted to experiment with representing a matrix as a dictionary and encapsulate it as a `Matrix` class. The class should:

1. represent the matrix as a dictionary.
2. accept two arguments (the number of rows and the number of columns) in the constructor. The elements in the initial matrix are all zeros.
3. support `get` an element of the matrix given its row and column as a tuple.
4. support `insert` of an element of the matrix given its row and column as a tuple. If the element already existed, the operation overrides the old entry.
5. support `delete` of an element of the matrix given its row and column as a tuple. We can assume such an element always exists when we want to do the deletion.
6. include a method named `dict2list` to convert the dictionary representation here to a usual matrix represented as a list (that explicitly representing all the elements in the matrix inclusive of any zero).
7. include a method named `transpose` to transpose the matrix to output a new matrix. Note that the operation does not change itself.
8. include a method named `multiply` which performs a matrix multiplication of itself with another input matrix (from this same class) to output a new matrix. Note that the operation does not change itself. We assume the input matrix is of the correct number of rows and columns to allow this matrix-matrix multiplication.

Here are some sample executions:

```
>>> m1 = Matrix(1, 3)
>>> m1.insert((0,0), 1)
>>> m1.insert((0,1), 2)
>>> m1.insert((0,2), 3)
>>> print(m1.dict2list())
[[1, 2, 3]]
>>> m1.delete((0,1))
>>> print(m1.dict2list())
[[1, 0, 3]]
>>> m2 = m1.transpose()
>>> print(m2.dict2list())
[[1], [0], [3]]
>>> m3 = Matrix(1, 4)
>>> m3.insert((0,0), 3)
>>> m3.insert((0,1), 4)
>>> m3.insert((0,3), 5)
>>> m4 = m2.multiply(m3)
>>> print(m4.dict2list())
[[3, 4, 0, 5], [0, 0, 0, 0], [9, 12, 0, 15]]
```

Use the above sample executions as guide to know about the correct interface to the various functions. Please develop the matrix class using dictionary as stated in the above.

Question 12.

A. The constructor and the methods `get`, `insert`, `delete`, and `dict2list`. [4 marks]

B. The method `transpose` as stated. [2 marks]

C. The method `multiply` to perform multiplication as stated. [2 marks]

Having tried out the above (with your help, of course), Violet realizes there is no particular advantage here with using dictionary for a matrix class though it was a good exercise to learn about multiple representations too – maybe other possibilities should be investigated in the coming time...

For now, you deserve a good rest for the rest of the day and forget about matrix for the time being - but not for too long - ”See you in two weeks’ time!”, Matrix said.

— E N D O F P A P E R —