CS1010X — Programming Methodology
National University of Singapore

Practical Examination

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**.
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 **3** questions printed in **6** 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 attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
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, 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. We have allocated small credit to grade the style and use of data structure.

GOOD LUCK!

Question 1 : Day of the Week [7 marks]

Now you can find out the day of the week, for example, of your birthday (which happened more than 20 years ago) – you can impress your family with that. Well, not interesting? How about finding out whether the upcoming public holiday (National Day, 9 August 2021) is a long weekend (without referring to a calendar)? etc.

Here is how to do it: you need to find out the numbers in the following 5 so-called **boxes**. Then, find out the remainder of the sum of these 5 boxes when divided by 7. The remainder of 0 means Sunday; 1 means Monday, 2 means Tuesday, 3 means Wednesday, 4 means Thursday, 5 means Friday, and 6 means Saturday.

Box 1 : This is simply the day of the month, e.g. for 5 June 2021, this box is 5.

Box 2 : This is a code assigned based on the month of the date as follows. For example, the code is 4 for 5 June 2021 (non-leap year) from the following tables. (Note: leap year is a year which is divisible by 400, or a year divisible by 4 but not by 100.)

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	3	3	6	1	4	6	2	5	0	3	5

Table 1: Code of Month for a non-leap Year

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
6	2	3	6	1	4	6	2	5	0	3	5

Table 2: Code of Month for a leap Year

Box 3 : This is a code of the century of the date. For example, 5 June 2021 is in the 21^{st} century and the code is 6 from the following table. And, for 5 June 1999 is in the 20^{th} century and the code is 0.

16^{th} century	17^{th} century	18^{th} century	19^{th} century	20^{th} century	21^{st} century
0	6	4	2	0	6

Table 3: Code of Century

Box 4 : This contains the last 2 digits of the year in the date. For example, this box is 21 for 5 June 2021, and 99 for 5 June 1999.

Box 5 : This box contains the integer division of Box 4 by 4. For 5 June 2021, we have 21 (in Box 4) that integer divide by 4 to give 5.

To complete our example of 5 June 2021, we have: 5 + 4 + 6 + 21 + 5 = 41 which has remainder 6 when divided by 7 and it is thus a Saturday.

Please pay attention to use the appropriate data structures of tuple vs list vs dictionary, and to use meaningful names for your variables in the above. We have allocated 2 marks (out of 7) to grade on these aspects.

Question 2 : Picture Language [12 marks]

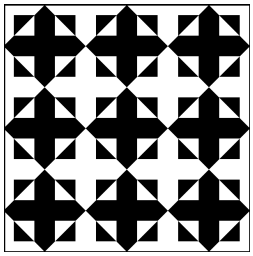


Figure 1: `show(nxn(3, make_cross(rcross_bb)))`

We have tried picture language in our lecture, mission and sidequest. Here is another exercise for you to show off what you know. You have seen the codes to create the above figure of a 3-by-3 pattern of crosses. This is done with the following codes as in our Lecture Note 2.

```
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))))

show(nxn(3, make_cross(rcross_bb)))
```

Notice that the result is done through two parts. First, it has the function `stackn` to create a column of crosses, and then use that in function `nxn` to create the needed result. For your reference, the result of `stackn(3, make_cross(rcross_bb))` is:

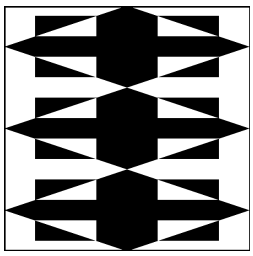


Figure 2: `show(stackn(3, make_cross(rcross_bb)))`

Now we want to create a variant of this feature with 2 pictures instead of 1, i.e. we want the following figure with `nxn_alt(5, make_cross(nova_bb), make_cross(rcross_bb))`:

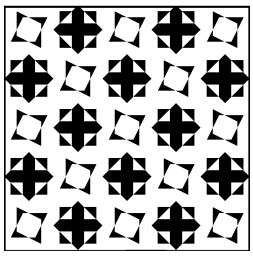


Figure 3: `show(nxn_alt(5, make_cross(nova_bb), make_cross(rcross_bb)))`

Following what we have learned, we first need a function `stackn_alt` to create a column of alternate picture 1 and picture 2, with picture 1 appears on the top. For example, `stackn_alt(5, make_cross(nova_bb), make_cross(rcross_bb))` is:

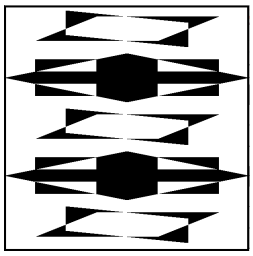
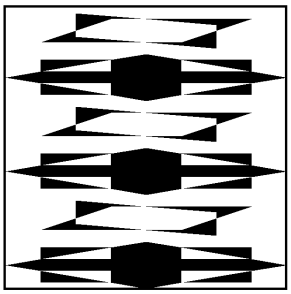


Figure 4: `show(stackn_alt(5, make_cross(nova_bb), make_cross(rcross_bb))).`

Below is another example with a stack of 6 alternate picture 1 and picture 2 on the left, and the resulting `nxn_alt` figure.



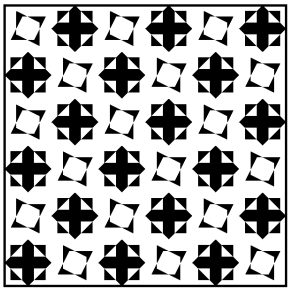


Figure 5: Left is `stackn_alt(6, make_cross(nova_bb), make_cross(rcross_bb))`, and the right is `nxn_alt(6, make_cross(nova_bb), make_cross(rcross_bb))`.

To sum up, you are to do the following (1 out of 4 marks in each part is used to grade your programming style):

A. Write a recursive version of `stackn_alt(n, pic1, pic2)`. [4 marks]

B. Following the codes of the function `nxn` given in the previous page, write a version of `nxn_alt(n, pic1, pic2)` in one line of `stackn_alt`. [4 marks]

C. Write an iterative version of `nxn_alt(n, pic1, pic2)` that uses `for` or `while` loop. [4 marks]

Question 3 : Rectangular Sum [13 marks]

Given a list of lists, called `lst`, we are interested to compute another list of lists, called `result_lst` whose entry $[i][j]$ is the sum of entries $[i'][j']$ in `lst` where $0 \leq i' \leq i$ and $0 \leq j' \leq j$. That is,

$$\texttt{result\_lst}[i][j] = \sum_{0 \leq i' \leq i, 0 \leq j' \leq i} \texttt{lst}[i'[j'] \tag{1}$$

For example,

```
lst = [ [1, 2, 3, 4, 5, 6, 7, 8],
        [9, 0, 1, 2, 3, 4, 5, 6],
        [7, 8, 9, 0, 1, 2, 3, 4],
        [5, 6, 7, 8, 9, 0, 1, 2],
        [3, 4, 5, 6, 7, 8, 9, 0] ]
```

then, we want to have (after nicely formatted):

```
result_lst = [ [1,   3,  6, 10,  15,  21,  28,  36],
               [10, 12, 16, 22,  30,  40,  52,  66],
               [17, 27, 40, 46,  55,  67,  82, 100],
               [22, 38, 58, 72,  90, 102, 118, 138],
               [25, 45, 70, 90, 115, 135, 160, 180] ]
```

The above can be achieved naively with the following Python codes:

```
def rect_sum(lst, i, j):
    sum = 0
    for ii in range(0, i+1):
        for jj in range(0, j+1):
            sum += lst[ii][jj]
    return sum

def naive_sum(lst):
    result_lst = []
    for i in range(0, len(lst)):
        result_lst.append([0]*len(lst[0]))
        for j in range(0, len(lst[0])):
            result_lst[i][j] = rect_sum(lst, i, j)
    return result_lst
```

The function `naive_sum` calls the function `rect_sum` to give the value for `result_lst`$[i][j]$. Apparently, this is naive as `rect_sum` repeated some earlier work done for smaller i and j to obtain results for larger i and j. We are to do something better in the following parts.

A. What is the time complexity of `naive_sum` with input of a list of lists, called `lst` of size $n \times m$ where n is `len(lst)` and m is `len(lst[0])`? [1 marks]

B. With some thinking, you will realize that this is once again a prefix sum in disguise. First, you can do a prefix sum "horizontally", followed by "vertically" to arrive at the same answer too. For our above example of `lst`, after the prefix sum done "horizontally", we get:

```
horizontal_lst = [ [1,  3,  6, 10, 15, 21, 28, 36],
                   [9,  9, 10, 12, 15, 19, 24, 30],
                   [7, 15, 24, 24, 25, 27, 30, 34],
                   [5, 11, 18, 26, 35, 35, 36, 38],
                   [3,  7, 12, 18, 25, 33, 42, 42] ]
```

The above `horizontal_lst` when subject to a prefix sum "vertically" is the `result_lst`. For example, the prefix sum of the second "column" (that is, `horizontal_lst[i][1]` for `i` from 0 to 4) of [3, 9, 15, 11, 7] is [3, 12, 27, 38, 45] as required.

Base on the above idea, you write a function called `better_sum` to return `result_lst` as required. That is, you will have a faster codes to produce the same output as that of `naive_sum`. 1 out of 4 marks is used to grade your programming style. [4 marks]

C. Interestingly, the same problem can be solved with the dynamic programming. That is, the value of `result_lst[i][j]` is actually a function of a few `result_lst[i'][j']` values with `i' < i` and `j' < j`, plus `lst[i][j]`. By drawing some rectangles in your question papers, you can figure out what is this function (formula) that you should have. Then, work out the Python codes for the function `dp_sum` with this dynamic programming approach. 1 out of 4 marks is used to grade your programming style. [4 marks]

D. With the previous two parts of `better_sum` and `dp_sum`, which one is better than the other, or there is no difference in terms of time complexity? State your reasoning. [1 marks]

E. Actually, `result_lst` not only tell us the sum of a rectangle from `[0][0]` to `[i][j]`, we can use it to tell us the sum of any rectangle from `[ii][jj]` to `[i][j]`, assuming `ii` $\leq$ `i` and `jj` $\leq$ `j`. That is, we can write an O(1) time function `search_rect_sum` with inputs of `result_lst, ii, jj, i, j`. Sample runs as follows:

```
>>> search_rect_sum(result_lst, 0, 0, 4,7)
180
>>> search_rect_sum(result_lst, 3, 6, 4,7)
12
>>> search_rect_sum(result_lst, 1, 0, 3, 6)
90
>>> search_rect_sum(result_lst, 1, 1, 3, 6)
69
>>> search_rect_sum(result_lst, 2, 3, 3, 6)
24
```

As you have learned from some previous part(s), you can figure out the formula to use to write the function `search_rect_sum` to achieve O(1) time complexity. You can assume there is no error in the inputs, in particular, `ii` $\leq$ `i` and `jj` $\leq$ `j` are always true. 1 out of 3 marks is used to grade your programming style. [3 marks]

— E N D O F P A P E R —