CS1010X — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

25 March 2023 **Time allowed:** 2 hours

Student No:	**S**	**O**	**L**	**U**	**T**	**I**	**O**	**N**	**S**

Instructions (please read carefully):

1. Write down your student number on the **question paper**. DO NOT WRITE YOUR NAME ON THE QUESTION PAPER!
2. This is **an open-book test**.
3. Please switch off your mobile phone, and no laptop and no calculator. If found using any of these equipment any moment from now till your scripts have been collected, you will be given a zero mark.
4. This paper comprises **FIVE (5) questions** and **TWELVE (12) pages**. The time allowed for solving this test is **2 hours**.
5. The maximum score of this test is **32 marks**. The weight of each question is given in square brackets beside the question number.
6. All questions must be answered in the space provided in this question paper. If you write something important as your answer outside the space provided, please make clear with some marking. At the end of the test, please submit your question paper only. No extra sheets will be accepted as answers.
7. You are allowed to use pencils. Please be sure your handwriting is legible and neat, and you have proper indentation as needed for your Python codes to avoid misinterpretation.

GOOD LUCK!

Question	Marks
Q1	
Q2	
Q3	
Q4	
Q5	
Total	

Question 1: Warming Up [6 marks]

A. For the following Python code, write the output of the `print` statement. [2 marks]

```
def f(a, b):
    a = 10
    def g(a, c):
        return a + b + c
    return g

print( f(11,9)(1,8) )
```

18

B. For the following Python code, write the outputs (2 lines in total) of the `print` statements. [2 marks]

```
t = (1, 2, 3)
s = ("a", "b", "c")
def fn(t, s):
    a = t
    b = s
    t = ( a, b )
    s = [ t ]
    return s

t = fn(s, t)
print( t )
print( s )
```

[(('a', 'b', 'c'), (1, 2, 3))]
('a', 'b', 'c')

C. Study the following codes. [2 marks]

```
def g(n, m):
    if n > m:
        return n * m
    else:
        return f(m, n)

def f(n, m):
    if n < m or m == 0:
        return m * n
    else:
        return f(n%m, m)
```

What can you claim about the outputs of the function `g(n, m)` when the inputs `n, m` are positive integers with `0 < n < m` ? Please first give your output in terms of the original inputs `n` and `m`, and then explain your reasoning.

The output is always: $(m\%n) * n$ where m and n are the original inputs.

This is because g will call f with m and n interchanged. Inside f, we have $n > m$, thus we call f recursively (under the else-part) with $(n\%m, m)$ to have $(n\%m) * m$ as the output (and the program stops) where n and m have been interchanged from the original inputs.

Question 2: Something familiar [6 marks]

A. The following function `bigger_sum` is supposed to take in three numbers and return the sum of the squares of the two larger numbers. [2 marks]

```
def sum_of_squares(a, b):
    return a * a + b * b

def bigger_sum(a, b, c):
    if a < b and a < c:
        return sum_of_squares(b, c)
    elif b < a and b < c:
        return sum_of_squares(a, c)
    else:
        return sum_of_squares(a, b)
```

Suppose the inputs `a, b, c` to the function `bigger_sum` are valid (and there is no syntax error for the codes), state whether the above implementation actually can or cannot produce the correct output for the stated objective. If can, please explain the logic of the program; if cannot, please provide a simple counter-example to show that the output is wrong.

This is in our tutorial 1.
The implementation is incorrect. It does not work for a=2, b=2, c=3. That is, the if-elif-else statement does not take care of when a==b.

Notes:
The question asks for "simple counter-example". This means really an example to be given – that is, explicit numbers for a, for b, and for c. Some of you wrote in general about a, b, and c. As long as the argument can still get the wrong answer (not meeting the objective), credit is given. But, it should be clear that saying the program does not work when there are 2 same numbers is not enough. For example, the program still produces correct output for a=1, b=2, c=2.

"Valid" here should not be interpreted as all numbers (a, b, c) must be distinct, and must fit the program. We write program to fit our objective, and not finding inputs to fit our program. So, any assumption on the inputs (such as all numbers must be distinct) is not accepted.

B. Given the below codes, you are to write out the output of each given `print` statement. you do not need to calculate out the number explicitly. That is, you can leave your answer as the power of some other number (that we can enter into a simple calculator to get the answer too).

```
def addF(f, g):
    return lambda x: f(x) + g(x)

def compose(f, g):
    return lambda x: f(g(x))

x = lambda x: x*x
y = lambda y: y**y


print( compose(x,y)(3) )
```

[2 marks]

729

```
print( compose( addF(x, y), addF(addF(y, x), x)) (2) )
```

[2 marks]

8 916 100 448 400 (or just leave it as: 12x12 + pow(12,12))

Question 3: Not really unfamiliar [7 marks]

Study the following codes carefully.

```
def create(t, m):
    n = len(t)
    while n > 1:
        s = ()
        for i in range(0, n, m):
            minX = t[i]
            for j in range(i+1, min(i+m, n)):
                if t[j] < minX:
                    minX = t[j]
            s = s + (minX,)
        n = len(s)
        t = s
        print(t)
    return t
```

Give the output of the following `print` statement. Your answer should have 4 separate lines. [2 marks]

```
print(create( (11,9,10,6,7,8,5,4,3,2,1), 3 ))
```

```
(9, 6, 3, 1)
(3, 1)
(1,)
(1,)
```

Note: The code is to consider segments of length m to find the minimum among the m elements to be kept into s. So, after one round (the inner two for-loops), the length of the tuple is about n/m, then after two rounds, the length of the tuple is about n/m^2, etc.

With the above tracing of the `print` statement, we want to understand the time and space complexity of the codes in terms of the length `n` of the input tuple `t`. Note that the input `m` is always a positive integer between 2 and 9, and `n` is much larger than `m`.

What is the space requirement of the codes? State your answer first, and then provide your reasoning. [2 marks]

O(n), as the codes use only a new tuple s with length less than n.

What is the time requirement of the codes? State your answer first, and then provide your reasoning. [3 marks]

Ans: O(n^2), as m is a constant.

The two for-loops (i and j) take time linear to the length of `t`, which is $n + n/m + n/m^2 +$. This is less than $2n$.

So, the time taken is dominated by the statement `s = s + (minX,)`

The lengths of s in one looping of i are of 1, then 2, then 3, till n/m. We can see that this is O($(n/m)^2$).

Then, in the next looping of i, the lengths of s are of 1, then 2, then 3 till $n/(m^2)$. So, this is O$(n/m^2)^2$.

Then, in the next looping of i, the lengths of s are of 1, then 2, then 3 till $n/(m^3)$. So, this is O$((n/m^3)^2$.

So, the total is: $(n/m)^2 + (n/m^2)^2 + (n/m^3)^2 ...) < 2(n/m)^2$ which is still O(n^2) as m is some constant below 10, and the sum $1 + 1/m^2 + 1/m^4 + ...$ is less than 2.

Question 4: The Familiar Counting of Squares? [3 marks]

There was a counting squares in last year's midterm; we are going to use it to solve this year's question. Below is the repeat of what appeared in last year's midterm together with its one possible solution. You can skip the following to read directly our question after the program `num_sq_given_grid_of(n)`.

Given a square grid of size n by n, we want to count the total number of squares of size 1 by 1 (there are $n \times n$ of them), size 2 by 2 (depending on n), and so on until size of n by n (just 1 such big square).

For a 1 by 1 grid, there is just 1 square in the grid.

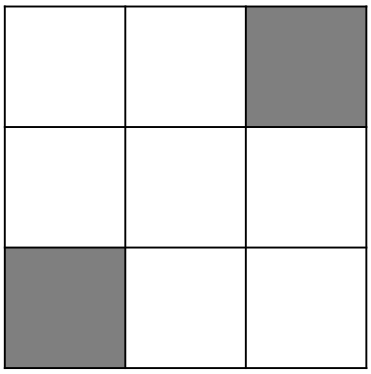
For a 2 by 2 grid, there are 4 squares of size 1 by 1, and 1 square of size 2 by 2, for a total of 5 squares in the grid.

For a 3 by 3 grid, there are 9 squares of size 1 by 1, 4 squares of size 2 by 2, and 1 square of size 3 by 3, for a total of 14 squares in the grid.

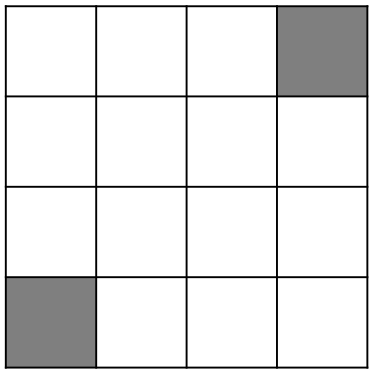
For a 4 by 4 grid, there are in total 30 squares we can find from the following Python codes.

```
def num_sq_given_grid_of(n):
    if n==1:
        return 1
    else:
        undercounted = 0
        for i in range(1, n+1):
            undercounted = undercounted + 2*(n-i)+1
        return undercounted + num_sq_given_grid_of(n-1)
```

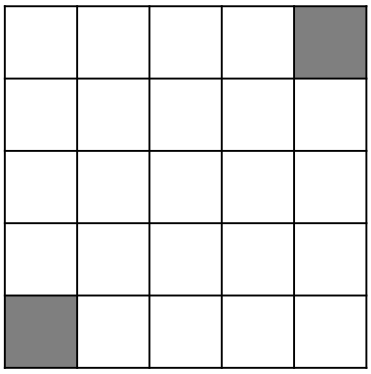
Our question this year is to count the number of squares for a $n \times n$ grid with a missing 1×1 grid at the lower left hand corner, and a missing 1×1 grid at the upper right hand corner. So, our grid is of size at least 3 by 3, then 4 by 4 and so on as follows:



3×3 grid



4×4 grid



5×5 grid

For a 3 by 3 grid with the two missing corners, there are 7 squares of size 1 by 1, 2 squares of size 2 by 2, and no square of larger sizes due to the missing corners.

For a 4 by 4 grid with the two missing corners, there are now 14 squares of size 1 by 1, 7 squares of size 2 by 2, and 2 squares of size 3 by 3.

For a 5 by 5 grid with the two missing corners, what is the total number of squares we can find? [1 marks]

46

Now you can write a program to do the count on the total number of squares for a grid of size n by n with the mentioned missing corners. Your program MUST make call(s) to the program `num_sq_given_grid_of` given in the above. That is, no credit will be given if you write a completely new (and correct) codes to solve the problem without using `num_sq_given_grid_of`. [2 marks]

```
def num_sq_given_grid_with_missing_corners (n):
#
# make sure you call num_sq_given_grid_of(n) in your answer
#

    if n < 3:        # or can put n==2, return 2
        return 0
    else:
        # we can use 2 times of (n-1)x(n-1) grid (without hole), but then
        # overcounted on the overlapping part of (n-2)x(n-2) grid
        #
        # other solution possible too....
        return 2*num_sq_given_grid_of(n-1) - num_sq_given_grid_of(n-2)
```

Question 5: A (slight) Different in Higher Order Function [10 marks]

You will be working with the following higher-order function `sumN` which is different from those you have encountered so far in our course material (such as `sum`).

```
def sumN(t, term, next, i, default):
    if t==() or i >= len(t) or i < 0:
        return default
    else:
        return term(t, i) + sumN(t, term, next, next(i), default)
```

A. The function `alt_sum_sq` takes an input tuple `t` (of numbers) to compute the alternating sum of the square of each term in `t`. That is, the output is:

$$\sum_{i=0}^{len(\mathrm{t})-1} (\mathrm{t}[i])^2 \cdot (-1)^i$$

Example execution:

```
>>> alt_sum_sq( (6,7,8) )   # = 6*6 - 7*7 + 8*8
51

>>> alt_sum_sq( (6,7,8,9) )   # = 6*6 - 7*7 + 8*8 - 9*9
-30
```

We can define `alt_sum_sq` with `sumN` as follows:

```
def alt_sum_sq( t ):
    return sumN( t, <T1>, <T2>, <T3>, <T4> )
```

Please provide possible lambda functions for `<T1>` and `<T2>`, and constants for `<T3>` and `<T4>`. We reserve the right to give no credit if your `<T1>` and `<T2>` do not match up to give a correct thinking to solve the problem. For example, no credit could be given when either `<T1>` or `<T2>` is left unanswered. [3 marks]

`<T1>`: [1 marks]

```
lambda t,i: t[i]*t[i]-t[i+1]*t[i+1] if i+1<=len(t)-1 else t[i]*t[i]
```

`<T2>`: [1 marks]

```
lambda x: x+2
```

`<T3>`: [0.5 marks]

```
0
```

`<T4>`: [0.5 marks]

```
0
```

What are the time and space complexities of your version of `alt_sum_sq` ? State your answers, and then provide your reasoning. No credit can be given if your `<T1>`, `<T2>`, `<T3>`, `<T4>` are far from correct. Also, we reserve the right to deduct some small credit should your time and space complexities are not matching up to the best that one can do with the given `sumN`; that is, your `<T1>`, `<T2>`, `<T3>`, `<T4>` may be correct but not as good. [2 marks]

Both time and space are O(n) where n is the length of the tuple t.

This is because we do not create new tuple in the recursive call in this case; we just stack up the recursive call that takes O(n) number of calls.

B. Here is your familiar `reverse(t)` function:

```
def reverse(t):
    if t == ():
        return ()
    else:
        return (t[-1],) + reverse(t[:-1])
```

We can actually define `reverse(t)` with `sumN` as follows:

```
def reverse( t ):
    return sumN( t, <T5>, <T6>, <T7>, <T8> )
```

Please provide possible lambda functions for `<T5>` and `<T6>`, and constants for `<T7>` and `<T8>`. We reserve the right to give no credit if your `<T5>` and `<T6>` do not match up to give a correct thinking to solve the problem. For example, no credit could be given when either `<T5>` or `<T6>` is left unanswered. [3 marks]

`<T5>`: [1 marks]

```
lambda t,i: (t[len(t)-i-1],)
```

`<T6>`: [1 marks]

```
lambda x: x+1
```

`<T7>`: [0.5 marks]

```
0
```

`<T8>`: [0.5 marks]

```
( )
```

What are the time and space complexities of your version of `reverse(t)`? State your answers, and then provide your reasoning. No credit can be given if your `<T5>`, `<T6>`, `<T7>`, `<T8>` are far from correct. Also, we reserve the right to deduct some small credit should your time and space complexities are not matching up to the best that one can do with the given `sumN`; that is, your `<T5>`, `<T6>`, `<T7>`, `<T8>` may be correct but not as good. [2 marks]

The time is O(n^2) where n is the length of the tuple t. But the space is actually O(n).

We need to create the tuple as we are getting "out" of the recursions (not when going "into" the recursion), and the time is now summing from 1 to n, which is O(n^2). But the space due to the construction of the tuples can be reused (as you can see in Python Tutor visualization) and thus O(n) is enough.

— END OF PAPER —