

# Midterm Test

30 March 2019

Time allowed: 1 hour 45 minutes

Student No:

S	O	L	U	T	I	O	N	S
---	---	---	---	---	---	---	---	---

## Instructions (please read carefully):

1. Write down your matriculation number on the **question paper**. DO NOT WRITE YOUR NAME ON THE QUESTION SET!
2. This is **an open-sheet test**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FIVE (5) questions** and **SEVENTEEN (17) pages**. The time allowed for solving this test is **1 hour 45 minutes**.
4. The maximum score of this test is **100 marks**. The weight of each question is given in square brackets beside the question number.
5. All questions must be answered correctly for the maximum score to be attained.
6. All questions must be answered in the space provided in the answer sheet; no extra sheets will be accepted as answers.
7. The back-sides of the sheets and the pages marked “scratch paper” in the question set may be used as scratch paper.
8. You are allowed to un-staple the sheets while you solve the questions. Please make sure you staple them back in the right order at the end of the test.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like (no red color, please).

# GOOD LUCK!

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Total		

## Question 1: Python Expressions [24 marks]

There are several parts to this problem. Answer each part **independently and separately**. In each part, one or more Python expressions are entered into the interpreter (Python shell). Determine the response printed by the interpreter for the final expression entered and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why. Partial marks may be awarded for workings if the final answer is wrong.

**A.**

```
a = 2
b = 1
def f(a, b):
    def g(a, b):
        if a > b:
            print("a > b")
        else:
            print("a !> b")
        b = a
    g(b, a)
f(b, a)
print("b is " + str(b))
```

[4 marks]

```
a > b
b is 1
```

Warm up: Tracing - a taste of scoping too.  
Quite a number of you forgot about "a > b" and thus lost 2 marks.

**B.**

```
num = "12345"
j=0
for i in num:
    j += 1
print(j, num in (num))
```

[4 marks]

```
5 True
```

num in (num) is a trick question. Did you fall for it?

One mark is deducted for "(5, True)" as the answer is not a tuple.

C.

```
def grade(score):
    if (score < 40):
        print("too bad: fail!")
    elif (score > 40):
        print("okay, just pass - work hard!")
    elif (score > 70):
        print("good, well done!")
    if (score > 80):
        print("Prof would be proud!")
    elif (score > 90):
        print("Excellent!")
grade(80.5)
```

[4 marks]

okay, just pass - work hard!  
Prof would be proud!

Test of if-then-else (without else is possible) and the order of condition is very important

D.

```
def small_table():
    for i in "0123":
        for j in range(1, int(i)):
            print(i + " x " + str(j) + " = " + i*j)
small_table()
```

[4 marks]

2 x 1 = 2  
3 x 1 = 3  
3 x 2 = 33

Test of iteration and range AND string multiplication.  
Two marks are deducted for not getting "3 x 2 = 33" correct.

**E.**

```
def generator(seed):
    def gen(bag, seed):
        if (seed == ""):
            if (bag==""):
                print("That's all")
            else:
                print(bag)
        else:
            gen(bag+seed[0], seed[1:])
            gen(bag, seed[1:])
    gen("", seed)
generator("123")
```

[4 marks]

```
123
12
13
1
23
2
3
That's all!
```

Test of recursion and slicing

**F.**

```
c = 5
def f(a):
    def g(a):
        d = a + c
        print(d)
    return g
f(3)(4)
```

[4 marks]

9

Test on handling a function that returns a function

**Question 2: Recurrence Relation [18 marks]**

Consider the sequence  $G$  generated with the following equation:

$$G(n) = G(n-1) + 2G(n-2) - 2G(n-3)$$

where  $G(0) = 1$ ,  $G(1) = 2$ , and  $G(2) = 3$ . The series generated by  $G(n)$ , starting with  $G(0)$  and so on is as follows:

1, 2, 3, 5, 7, 11, 15, 23, 31, ...

**A.** Provide a recursive implementation of  $G$  with an input  $n$ . [4 marks]

```
def G(n):
    if (n <= 2):
        return n+1
    else:
        return G(n-1) + 2*G(n-2) - 2*G(n-3)
```

**B.** What is the order of growth in time and in space for the function you wrote in Part A (in terms of  $n$ )? [2 marks]

Time:  $O(3^n)$

Space:  $O(n)$

These are marked according to what you did for the previous Part. So, marks are still awarded even though your algorithm is incorrect. Likewise, no mark is given for getting the same as our solutions here if they do not correspond to your given algorithm.

C. Provide an iterative implementation of  $G$  with an input  $n$ .

[4 marks]

```
def G(n):  
    if (n<=2):  
        return n+1  
    else:  
        m1, m2, m3 = 3, 2, 1  
        for i in range(3, n+1):  
            m1, m2, m3 = m1 + 2*m2 - 2*m3, m1, m2  
        return m1
```

D. What is the order of growth in time and in space for the function you wrote in Part C (in terms of  $n$ )?

[2 marks]

Time:  $O(n)$

Space:  $O(1)$

These are marked according to what you did for the previous Part. So, marks are still awarded even though your algorithm is incorrect. Likewise, no mark is given for getting the same as our solutions here if they do not correspond to your given algorithm.

Looking harder at the equation, you notice the following pattern...

$$\begin{aligned}
 G(n) &= G(n-1) + 2G(n-2) - 2G(n-3) \\
 G(n-1) &= G(n-2) + 2G(n-3) - 2G(n-4) \\
 G(n-2) &= G(n-3) + 2G(n-4) - 2G(n-5) \\
 &\vdots \\
 G(3) &= G(2) + 2G(1) - 2G(0)
 \end{aligned}$$

**E.** Provide a recursive implementation of  $G$  for input  $n$  that is significantly faster than what you wrote in Part A above. [4 marks]

```
def G(n):
    if (n<=2):
        return n+1
    else:
        return 2*G(n-2) + 1 # 2*G(n-2) + G(2) - 2 G(0)
```

**F.** What is the order of growth in time and in space for the function you wrote in Part E (in terms of  $n$ ). [2 marks]

Time:  $O(n)$

Space:  $O(n)$

These are marked according to what you did for the previous Part. So, marks are still awarded even though your algorithm is incorrect. Likewise, no mark is given for getting the same as our solutions here if they do not correspond to your given algorithm.

### Question 3: Higher Order Functions [24 marks]

A. In this question, you will define a few versions of a function `sum_cross_fib` with the following higher-order functions:

```
def fib(n):
    # fibonacci sequence is: 1, 1, 2, 3, 5, 8, 11, 19, ...
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def sum(term, a, next, b):
    if a > b:
        return 0
    else:
        return term(a) + sum(term, next(a), next, b)

def back_sum(term, a, back, b):
    if a > b:
        return 0
    else:
        return term(b) + back_sum(term, a, back, back(b))

def fold(op, f, n):
    if n==0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))
```

The function `sum_cross_fib` takes an input tuple  $\tau$  (of numbers) to compute the sum of the product of each  $i^{th}$  number in  $\tau$  with the corresponding  $i^{th}$  fibonacci number:

$$\text{sum\_cross\_fib}(\tau) = \sum_{i=0}^{\text{len}(\tau)-1} \tau[i] \times \text{fib}(i)$$

Example execution:

```
>>> sum_cross_fib((1,2,3)) # 1x1 + 2x1 + 3x2 = 9
9
```

```
>>> sum_cross_fib((5, 4, 3, 2, 1)) # 5x1 + 4x1 + 3x2 + 2x3 + 1x5 = 26
26
```



Firstly, we define it with `sum` as follows:

```
def sum_cross_fib(t):
    return sum(<T1>, <T2>, <T3>, <T4>)
```

Please provide possible implementations for T1, T2, T3, and T4.

[6 marks]

<T1>:  
[2 marks] `lambda a : fib(a) * t[a]`

<T2>:  
[1 marks] `0`

<T3>:  
[2 marks] `lambda n: n + 1`

<T4>:  
[1 marks] `len(t) - 1`

**B.** Secondly, we define it with `back_sum` as follows:

```
def sum_cross_fib(t):
    return back_sum(<T5>, <T6>, <T7>, <T8>)
```

Please provide possible implementations for T5, T6, T7, and T8.

[6 marks]

<T5>:  
[2 marks] `lambda a : fib(a) * t[a]`

<T6>:  
[1 marks] `0`

<T7>:  
[2 marks] `lambda n: n - 1`

<T8>:  
[1 marks] `len(t) - 1`

C. Thirdly, we define it with `fold` as follows:

```
def sum_cross_fib(t):
    return fold(<T9>, <T10>, <T11>)
```

Please provide possible implementations for T9, T10, and T11.

[5 marks]

<p>&lt;T9&gt;: [2 marks]</p>	<p>lambda x,y: x + y</p>
<p>&lt;T10&gt;: [2 marks]</p>	<p>lambda n: t[n] * fib(n)</p>
<p>&lt;T11&gt;: [1 marks]</p>	<p>len(t) - 1</p>

D. We next consider another function called `cross_tuples` that takes as input two equal length tuples  $\tau_1$  and  $\tau_2$  (of numbers) to compute the product of their corresponding numbers:

$$\text{cross\_tuples}(\tau_1, \tau_2) = (\tau_1[0] \times \tau_2[0], \tau_1[1] \times \tau_2[1], \dots, \tau_1[\text{len}(\tau_1) - 1] \times \tau_2[\text{len}(\tau_2) - 1])$$

Example execution:

```
>>> t1 = (5, 4, 3, 2, 1, 4, 6)
>>> t2 = (1, 2, 3, 4, 5, 6, 7)
>>> cross_tuples(t1, t2)
(5, 8, 9, 8, 5, 24, 42)
```

Define `cross_tuples` in terms of one of `sum`, `back_sum` or `fold` (pick a one will do). For example, perhaps

```
def cross_tuples(t1, t2):
    return sum(<T12>, <T13>, <T14>, <T15>)
```

where T12, T13, T14 and T15 are appropriate expressions. Or, perhaps not? In other words, the answer should be just one line of code.

[7 marks]

```
def cross_tuples(t1, t2): # assume t1 and t2 are of the same length
    return fold(lambda x, y: y + x, lambda n: (t1[n]*t2[n],), len(t1)-1)
```

1 mark: correctly using `fold` in your answer, else 0 for the whole question as you are asked to use only `sum`, `back_sum` or `fold`.

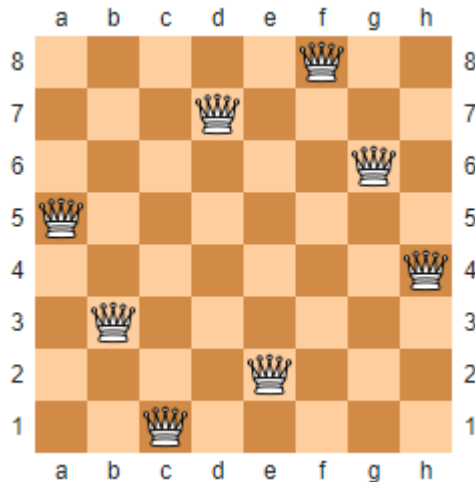
2 marks: correctly reverse x and y, else 1 mark for correct op for addition

2 marks: correctly create tuple for function f, else 1 or less

2 marks: correctly put in n

### Question 4: N Queens Problem [31 marks]

Chess composer Max Bezzel published the eight queens puzzle in 1848. Franz Nauck published the first solutions in 1850. The eight queens puzzle is the problem of placing eight chess queens on an  $8 \times 8$  chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. One solution to the eight queens problem is shown below. (Source: Wikipedia)



In this problem, you will design the chessboard required to solve a more general  $n$ -queens problem. In particular, you will implement the following functions:

1. `make_board` takes in an integer  $n > 0$  and creates an empty  $n \times n$  chessboard.
2. `add_queen` takes a board, a row and a column and returns a new board with a new queen at the specified position. The indices start at 0. If there is already a queen in the specified position, this function will return `False`.
3. `num_queens` takes a board and returns the number of queens on the board.
4. `size` takes a board and returns the width (or length) of the board.

You are advised to read through all the requirements for this question carefully before deciding on the implementation for your board object.

**A.** Decide on an implementation for the board object and implement `make_board`. Describe how the state is stored in your implementation and explain how you will track changes to the board. [5 marks]

**Note:** You are limited to using tuples for this question, i.e. you cannot use lists and other Python data structures.

There are many possibilities. Here, we just store the state of the board as an  $n \times n$  2-dimensional array implemented with tuples. We use 0 to indicate an empty position and a 1 to indicate a queen. This makes it easy to do computations. Clearly a student can use "Q" to represent the queen, but that makes life harder.

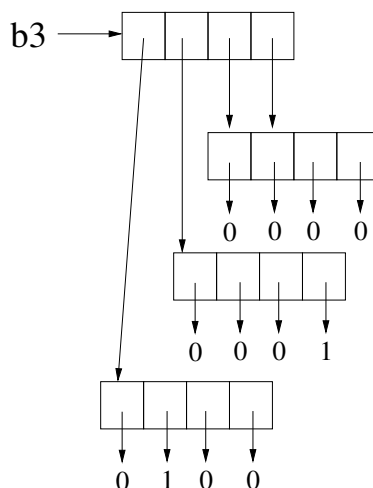
```
def make_board(n):
    return ((0,) * n,) * n
```

We are testing that the students know how to multiply tuples correctly.

Consider the following sample execution:

```
>>> b1 = make_board(4)
>>> b2 = add_queen(b1, 0, 1)
>>> b3 = add_queen(b2, 1, 3)
```

**B.** Draw the box-and-pointer diagram for your implementation corresponding to the board b3. [4 marks]



-2 if tuple multiplication was used in Part (A) but sharing is not correct in the diagram.  
 -2 if anyhow draw box-and-pointer diagram. :'(

**C.** Implement the function `add_queen` that takes a board, a row and a column and returns a new board with a new queen at the specified position. The indices start at 0. If there is already a queen in the specified position, `add_queen` will return `False`. [5 marks]

```
def add_queen(b, r, c)
    if b[r][c] == 1:
        return False
    else:
        new_row = b[r][:c] + (1,) + b[r][c+1:]
        return b[:r] + (new_row,) + b[r+1:]
```

-2 marks if forgot to check if queen is already there.

**D.** Implement the function `num_queens` that takes a board and returns the number of queens on the board. [3 marks]

```
def num_queens(b):
    total = 0
    for row in b:
        total += sum(row)
    return total
```

**E.** Implement the function `size` that takes a board and returns the width (or length) of the board. [2 marks]

```
def size(b):
    return len(b)
```

**F.** Implement the function `simple_solved` that takes a board and returns `True` if there are  $n$  queens and none of the queens threaten each other either vertically or horizontally. Or `False`, otherwise. Note that this checks if the board partially solves the N Queens problem. [6 marks]

**Example execution:**

```
>>> b1 = make_board(4)
>>> b2 = add_queen(b1, 0, 1)
>>> b3 = add_queen(b2, 1, 3)
>>> b4 = add_queen(b3, 2, 2)
>>> b5 = add_queen(b4, 3, 0)
>>> simple_solved(b5)
True
```

```
def simple_solved(b):
    #check rows
    for row in b:
        if sum(row) != 1:
            return False

    #check cols
    for i in range(size(b)):
        total = 0
        for row in b:
            total += row[i]
        if total != 1:
            return False

    return True
```

We are just testing that the student knows how to write the right `for` loops in this problem. Not that if every row has exactly one queen then we know that there are exactly  $n$  queens.

**G.** Implement the function `is_solved` that takes a board and returns `True` if the board solves the N Queens problem, i.e. no two queens share the same row, column, or diagonal. Or `False`, otherwise. [6 marks]

**Example execution:**

```
>>> b1 = make_board(4)
>>> b2 = add_queen(b1, 0, 1)
>>> b3 = add_queen(b2, 1, 3)
>>> b4 = add_queen(b3, 2, 2)
>>> b5 = add_queen(b4, 3, 0)
>>> is_solved(b5)
False
```

```
>>> b6 = add_queen(b3, 2, 0)
>>> b7 = add_queen(b6, 3, 2)
>>> is_solved(b7)
True
```

```
def is_solved(b):
    if not simple_solved(b):
        return False

    for i in range(1, 2*size(b)-1):
        total = 0
        for j in range(size(b)):
            for k in range(size(b)):
                if j+k == i:
                    total += b[j][k]
            if total > 1:
                return False

    for i in range(-size(b)+1, size(b)):
        total = 0
        for j in range(size(b)):
            for k in range(size(b)):
                if j-k == i:
                    total += b[j][k]
            if total > 1:
                return False

    return True
```

2 points for `simple_solved`.

The solution requires the insight that the diagonals will satisfy either  $i + j = p$ , where  $0 \leq p < 2n$  or  $i - j = q$ , where  $-n < q < n$ . In other words, need to check whether the gradients between any 2 pairs of coordinates are 1 or -1.

**Question 5: Why Are You Here? [3 marks]**

You have successfully made it to college. What do you hope to get out of your NUS education? How do you think what you have learnt in CS1010X will help you achieve that goal? Explain.

Student will get points for any reasonably attempt that demonstrates effort.  
3/3 if you make you prof laugh out loud.

If student writes random stuff, then roughly 1 point is given for each “reasonable” point.



Scratch Paper

— END OF PAPER —