CS1010X — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

31 March 2018 **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 **NINETEEN (19) 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.

```
x = 3
y = 5
def f(x,y):
    x = 7
    def g(x):
        return y
    if x > y:
        print(f(y,x))
    else:
        print(f(y-x,x))
f(y,x)
```

[4 marks]

Infinite loop. This is to test that the student can trace this code correctly.

B.

```
x = 5
y = 7
if y//x>=1:
    print("I")
if (x+y)%5>2:
    print("love")
elif (y**2)%4==2:
    print("CS1010X")
else:
    print("midterms")
```

[4 marks]

```
I
midterms
```

This is to test that students understand `if-elif` and also the modulus operators.

C.

```
result = ()
for i in range(6):
    if i%2==1:
        result = result + (i,)
    else:
        result = (result)*2
        i = i+3
print(result)
```

[4 marks]

```
(1, 1, 3, 1, 1, 3, 5)
```

This is to test that students understand and can trace through a `for` loop with iterating through a tuple. Also tests that student knows that `i=i+3` does nothing inside the `for` loop.

D.

```
def dozo(x):
    return 2*x if x else "Cool"

print(dozo(1))
print(dozo("False"))
print(dozo(()))
print(()==False)
```

[4 marks]

```
2
FalseFalse
Cool
False
```

This is to test that students understand the ternary `if-else` form and also that `()` is interpreted as `False` in a conditional statement but is not `False`.

E.

```
a = 5
b = 3
def foo(a):
    def b(a):
        return a**3
    return b
print(foo(a)(b))
```

[4 marks]

27

This is to test that the students understand how to handle a function that returns a function and also overriding of variables.

F.

```
a = "holy"
b = "cow"
def potato(x,y):
    result = ""
    for i in x:
        for j in y:
            if i>j:
                result += i
    return result
c = potato(a,b)
print(c[3:25])
```

[4 marks]

yyy

This is to test that the students understand iteration over strings and how to compare strings and also that if slicing goes beyond range, there is no error.

Question 2: Fibonacci Sum [18 marks]

By now, you should be familiar with the Fibonacci series:

$$0, 1, 1, 2, 3, 5, 8, 13, \cdots$$

where $fib(0) = 0$, $fib(1) = 1$, and $fib(n) = fib(n-1) + fib(n-2)$ for $n \geq 1$,

A. [Warm Up] Provide an iterative implementation of `fib`. [2 marks]

```
def fib(n):
    a,b = 0,1
    for i in range(n):
        a,b = b,a+b
    return a
```

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

Time: $O(n)$

Space: $O(1)$

C. Consider the following sum

$$fib(0) + fib(1) + \cdots + fib(n)$$

which is the sum of the first $n+1$ Fibonacci numbers, which we will call `fib_sum`. Provide a **recursive** implementation of `fib_sum`. [3 marks]

```
def fib_sum(n):
    if n == 0:
        return fib(0)
    else:
        return fib(n) + fib_sum(n-1)
```

D. Assuming iterative `fib`, what is the order of growth in terms of time and space for the function you wrote in Part (C) in terms of n. Explain your answer. [2 marks]

Time: $O(n^2)$, since order of growth for `fib` is $O(n)$, so we will be adding $n + (n-1) + \cdots + 2 + 1$.

Space: $O(n)$, since the recursion will have n deferred operations and order of growth for `fib` is $O(1)$.

E. Provide an **iterative** implementation of `fib_sum`. [3 marks]

```
def fib_sum(n):
    result = 0
    for i in range(n+1):
        result += fib(i)
    return result
```

F. Assuming iterative `fib`, what is the order of growth in terms of time and space for the function you wrote in Part (E) in terms of n. Explain your answer. [2 marks]

Time: $O(n^2)$, since order of growth for `fib` is $O(n)$, so we will be adding $n+(n-1)+\cdots+2+1$.

Space: $O(1)$, since only 2 variables needed here and order of growth for `fib` is $O(1)$.

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

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

G. Provide an implementation of `fib_sum` that is significantly faster than what you wrote in Parts (C) and (E) above. [2 marks]

```
def fib_sum(n):
    return fib(n+2) - 1
```

H. Assuming iterative `fib`, what is the order of growth in terms of time and space for the function you wrote in Part (G) in terms of n. Explain your answer. [2 marks]

Time: $O(n)$, since it's the same as `fib`!

Space: $O(1)$, since it's the same as `fib`!

Question 3: Higher Order Functions [25 marks]

A. We note the we can define `fib_sum(n)` from Question 2 in terms of `sum` (see Appendix) as follows:

```
def fib_sum(n):
    return sum(<T1>,<T2>,<T3>,<T4>)
```

Please provide possible implementations for T1, T2, T3, and T4. [6 marks]

`<T1>`: [2 marks]	`fib`
`<T2>`: [1 marks]	`0` or `1`
`<T3>`: [2 marks]	`lambda n:n+1`
`<T4>`: [1 marks]	`n`

B. We note the we can define `fib_sum(n)` from Question 2 in terms of `fold` (see Appendix) as follows:

```
def fib_sum(n):
    return fold(<T5>,<T6>,<T7>)
```

Please provide possible implementations for T5, T6, and T7. [5 marks]

`<T5>`: [2 marks]	`lambda x,y:x+y`
`<T6>`: [2 marks]	`fib`
`<T7>`: [1 marks]	`n`

Consider the following Python code:

```
def twice(f):
    return lambda x: f(f(x))

def thrice(f):
    return lambda x: f(f(f(x)))

def double(x):
    return 2*x

twice_double = twice(double)
thrice_double = thrice(double)
```

C. Provide an implementation of the function `repeated(f,n)` so that we would be able to define `twice_double` and `thrice_double` as follows:

```
twice_double = repeated(double,2)
thrice_double = repeated(double,3)
```

[6 marks]

```
def repeated(f,n):
    def helper(x):
        result = x
        for i in range(n):
            result = f(result)
        return result
    return helper
```

«««< midterm.tex

```
## Alternative solution
def repeated(f,n):
    if n==1:
        return f
    return lambda x: repeated(f,n-1)(f(x))
```

Partial credits were given if the input `x` is outside repeated application, e.g. `f(f(f))(x)`:

- (2 marks) if the code produced repeated application of `f` `n` times
- (1 mark) if the code produced repeated application of `f` but got the number wrong

=======

The following is an alternative recursion solution:

```
def repeated(f,n):
    if n==1:
        return f
    else:
        return lambda x: repeated(f,n-1)(f(x))
```

»»»> 1.22

Given 2 functions f and g, we want to construct the following composite function with a Python function `dual_function(f,g,n)`:

$$f(g(f(g(\cdots(x))\cdots)$$

where functions f and g are applied alternately n times in total, not each. The leftmost (outer) function is always f. In other words, `dual_function(f,g,4)` will produce:

```
lambda x: f(g(f(g(x))))
```

while `dual_function(f,g,7)` will produce:

```
lambda x: f(g(f(g(f(g(f(x)))))))
```

D. Provide an implementation of the function `dual_function(f,g,n)`. [6 marks]

```
def dual_function(f,g,n):
    def helper(x):
        f1,g1 = f,g
        if n%2==0:
            f1,g1 = g1,f1
        for i in range(n):
            x = f1(x)
            f1,g1= g1,f1
        return x
    return helper
```

This is a test of whether the students have fully understood `dual_fractal` from Mission 2.
Partial credits were given if the input `x` is outside repeated application, e.g. `f(g(f))(x)`:

- (2 marks) if the code produced alternating `f` and `g` applications `n` times and the outermost was always `f`
- (1 mark) if the code produced alternating `f` and `g` applications but the number was wrong or the outermost was not `f` sometimes

E. Define `repeated(f,n)` in terms of `dual_function(f,g,n)`. [2 marks]

```
def repeated(f,n):
    return dual_function(f,f,n)
```

Question 4: Room Booking System [30 marks]

Your company has a number of meeting rooms but everyone wants to use them and the current system involves pre-booking time slots for the meeting rooms on a sheet of paper that is stuck on the wall. Your boss found out that you are currently taking CS1010X and asks you to write a room booking system for your company. You decided that this is the opportunity to impress him with your coding skills. A record of the room bookings is called a schedule.

To fully satisfy the requirements for this question, you are advised to read through all the subquestions before you start. Your answer for Part(A) will affect your answer to subsequent sub-problems. If you do not make a wise decision in Part(A), you might end up having a really hard time with the later subquestions.

You are required to implement the following 6 functions in this problem:

1. `create_schedule` takes in a tuple of rooms (which are strings) and returns a new empty schedule for the rooms.
2. `is_schedule` takes a object and returns `True` if it is a valid schedule, or `False` otherwise.
3. `get_venues` returns a list of the rooms that is managed by the schedule.
4. `reserve` will take a room, a range of time slots and return a new schedule with the specified time slots reserved. Operation fails with `False` if one of the slots is already reserved.
5. `is_reserved` takes a room and time slot returns `True` if it is already reserved, or `False` otherwise.
6. `remove_reservation` returns a new schedule with the specified time slot removed.

Example execution:

```
>>> s1 = create_schedule(("LT15", "LT2", "LT19"))
>>> get_venues(s1)
('LT15', 'LT2', 'LT19')

>>> is_schedule(s1)
True
>>> is_schedule(())
False
>>> is_schedule("Mambo")
False

>>> is_reserved(s1, "LT15", 4)
False

>>> s2 = reserve(s1,"LT15",3,5)
>>> get_venues(s2)
('LT15', 'LT2', 'LT19')
>>> is_schedule(s2)
True
```

```
>>> is_reserved(s2, "LT15", 4)
True
>>> reserve(s1,"LT15",3,5)
False

>>> s3 = remove_reservation(s2, "LT15", 4)
>>> is_schedule(s3)
True
>>> is_reserved(s3, "LT15", 4)
False
```

A. Decide on an implementation for the schedule object and implement `create_schedule`. Describe how the state is stored in your implementation and explain. [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 store the state as a list of tuples where the first element is the name of the room. `def create_schedule(rooms):`

```
    result = ("#schedule#",)
    for room in rooms:
        result += ((room,),)
    return result
```

`"#schedule#"` is the label for the schedule object.

It was apparently that this question was not well-defined. Marks were given for any "reasonable" interpretation with attention focussed on the student' understanding of how to manipulate tuples correctly. For example, some students assumed that there were exactly 24 slots (for hours of the day). Always ask if you feel that there's ambiguity in a question.

B. Implement the function `is_schedule(s)` that returns `True` if `s` is a valid schedule, or `False` otherwise. [4 marks]

```
def is_schedule(obj):
    return type(obj)==tuple and obj != () and obj[0] == "#schedule#"
```

-2 marks for missing out `type(object) == tuple`.
-1 mark for `type(object) == tuple` at the end of an `and` statement (it neeeds to be the first thing!).
-2 marks for missing out a label, though this can be forgiven if the student defines a very special structure that can be tested.

C. Implement the function `get_venues(s)` that returns a tuple that contains the list of rooms for a schedule. [3 marks]

```
def get_venues(schedule):
    result = ()
    for booking in schedule[1:]:
        result += (booking[0],)
    return result
```

D. Implement the function `reserve(schedule, room, start, end)` that will return a new schedule where the specified room is reserved for time slots from *start* to *end* (inclusive). You can assume that both *start* and *end* are non-negative integers and that $start \leq end$. If some of the slots in the specified time slots are already reserved, return `False`. [7 marks]

```
def reserve(schedule, room, start, end):
    result = (schedule[0],)
    for booking in schedule[1:]:
        if booking[0] == room:
            slots = booking[1:]
            for i in range(start,end+1):
                if i not in booking[1:]:
                    slots += (i,)
                else:
                    return False
            result += ((booking[0],)+slots,)
        else:
            result += (booking,)
    return result
```

-2 marks if the student misses out one of the `(... ,)` in
`result += ((booking[0],)+slots,)`
-3 marks if the student tries to do assignment (=) because student does not understand that tuples are immutable.
-2 marks for failure to check in between start and stop (for certain implementations).

E. Implement the function `is_reserved(schedule, room, slot)` that returns `True` if it is already reserved, or `False` otherwise. [4 marks]

```
def is_reserved(schedule, room, slot):
    for booking in schedule[1:]:
        if booking[0] == room:
            return slot in booking[1:]
    return False
```

Some students came up with a v elegant solution and it looks like:

```
def is_reserved(schedule, room, slot):
    return reserve(schedule, room, slot, slot) == False
```

F. Implement the function `remove_reservation(schedule, room, slot)` that will return a new schedule where the specified is removed, or the original schedule if the slot to remove is not occupied. [7 marks]

```
def remove_reservation(schedule, room, slot):
    def remove_slot(booking, slot):
        if booking == ():
            return ()
        elif booking[0] == slot:
            return booking[1:]
        else:
            return (booking[0],) + remove_slot(booking[1:], slot)

    result = (schedule[0],)
    for booking in schedule[1:]:
        if booking[0] == room and slot in booking[1:]:
            result += (remove_slot(booking, slot),)
        else:
            result += (booking,)
    if result != schedule:
        return result
    else:
        return schedule
```

Key lesson here is to define a helper function to remove the slot. If not, this will get very hairy very rapidly.
-3 marks for failure to check in between start and stop (for certain implementations).
-2 marks for not returning the original tuple if there is no change.
Only 2 marks if mostly correct, but try to do assignment (=) because student does not understand that tuples are immutable.

Question 5: Draw Something! [3 marks]

You have been taking CS1010X for about 3 months now. Draw(!) something below that best expresses the most important thing(s) that you have learnt in the class thus far.

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.
Full marks for listing out 3 key concepts.

Appendix

The following are some functions that were introduced in class. For your reference, they are reproduced here.

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

def product(term, a, next, b):
  if a > b:
    return 1
  else:
    return term(a) * product(term, next(a), next, b)

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

def enumerate_interval(low, high):
    return tuple(range(low,high+1))

def filter(pred, seq):
    if seq == ():
        return ()
    elif pred(seq[0]):
        return (seq[0],) + filter(pred, seq[1:])
    else:
        return filter(pred, seq[1:])

def accumulate(fn, initial, seq):
    if seq == ():
        return initial
    else:
        return fn(seq[0], accumulate(fn, initial, seq[1:]))
```

Scratch Paper

— E N D O F P A P E R —