CS1010S — Programming Methodology
School of Computing
National University of Singapore

Solutions for Mid-Term Quiz

1 October 2014 **Time allowed:** 1 hour 45 minutes

Matriculation No:									

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 quiz**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FOUR (4) questions** and **NINETEEN (19) pages**. The time allowed for solving this quiz is **1 hour 45 minutes**.
4. The maximum score of this quiz 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 quiz.
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		
Total		

Question 1: Python Expressions [30 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. If the interpreter produces an error message, or enters an infinite loop, explain why.

A.

```
x = "0"
if x:
    print("Yeah")
elif not x:
    print("No!!")
else:
    print("No idea!")
```

[5 marks]

```
Yeah
```

This question tests if the student understands the `if` statement and assignment. It also tests understanding of what evaluates to false in Python.

B.

```
x = 3
y = 5
def f(x,y):
    x = 4
    return x + y
print(f(y,x))
```

[5 marks]

```
7
```

This question tests if the student understands how to evaluate a simple function call, and variable scoping.

C.
```
x = 1
for i in range(2,9):
    if i%2 == 0:
        x += i
    elif i>5:
        continue
    else:
        x -= 1
print(x)
```

[5 marks]

`19`

This question tests if the student understands the `if` statement, `for` loop, as well as `continue` and `break`

D.
```
x = 12345678
y = 0
while x > 0:
    y += 1
    x = x//10
print(x + y)
```

[5 marks]

`8`

This question tests if the student understands the `while` loop and integer division `//`.

E.
```
def foo(x):
    return lambda y: x(x(y))
print(foo(foo)(lambda x:x+1)(2))
```

[5 marks]

```
6
```

This question tests if the student understands how to evaluate complicated nested functions. Partial marks given if student shows intermediate evaluation of foo(foo) to lambda(y): foo(foo(y))

F.
```
def bar(x,y):
    return x + y
print(bar(((1,),),bar((1,2),(2,3))))
```

[5 marks]

```
((1,), 1, 2, 2, 3)
```

This question tests if the student understands tuples and how the + operator works on tuples. Partial marks given if i) the sequence 1,1,2,2,3 appears but brackets are wrong, or ii) (1, 2, 2, 3) was shown to be grouped in student’s workings.

Question 2: Alternating Series Galore [24 marks]

Consider the following alternating series s_{11}:

$$s_{11}(n) = 1 - 2 + 3 - 4 + \cdots n$$

A. **[Warm Up]** Write an **recursive** function `s11(n)` that returns the value for $s_{11}(n)$. [4 marks]

```
def s11(n):
    if n == 1:
       return 1
    elif n%2 == 0:
       return s11(n-1) - n
    else:
       return s11(n-1) + n
```

In this question and thereafter:
- full mark for a completely correct answer
- partial mark for a partially correct answer (e.g. -1 mark for incorrect base case, -1 for incorrect indentation, -1 for syntax errors)
- zero mark for a completely incorrect answer or no answer

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. Explain your answer. [4 marks]

Time: $O(n)$
There is a constant number of basic steps in each recursive call, and there are n such recursive calls from s11(n) to s11(1).

Space: $O(n)$
To compute s11(n), there is n-1 pending operations: s11(n-1), ..., s11(1)

-1 for each missing (or incomplete) explanation.
-1 for incorrect big-O notation.

C. Write an **iterative** function `s11(n)` that returns the value for $s_{11}(n)$. [4 marks]

```
def s11(n):
    total = 0
    for i in range(1,n+1):
       if i%2 == 0:
          total -= i
       else:
          total += i
    return total
```

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

Time: $O(n)$

Space: $O(1)$

Now, consider the following alternating series s_{ij} where $i, j \geq 1$:

$$
\begin{aligned}
s_{21}(n) &= 1+2-3+4+5-6+7+8-\cdots n \\
s_{12}(n) &= 1-2-3+4-5-6+7-8-\cdots n \\
s_{31}(n) &= 1+2+3-4+5+6+7-8+\cdots n
\end{aligned}
$$

Note that the subscript i denotes the number of terms with a positive sign and the subscript j denotes the number of terms with a negative sign. Basically these series will alternate between i positive terms and j negative terms.

E. Write a function `s21(n)` that returns the value for $s_{21}(n)$. [5 marks]

```
def s21(n):
    total = 0
    for i in range(1,n+1):
       if (i-1)%3 < 2:
          total += i
       else:
          total -= i
    return total
```

F. Write a function `make_s(i,j)` that returns the function $s_{ij}(n)$. In other words, we could have defined `s11(n)` for Part (A) (or (C) depending on your implementation) as follows:

```
s11 = make_s(1,1)
s12 = make_s(1,2)
```

[5 marks]

```
def make_s(i,j):
    def helper(n):
        total = 0
        for e in range(1,n+1):
            if (e-1)%(i+j) < i:
                total += e
            else:
                total -= e
        return total
    return helper
```

Question 3: Higher-Order Alternating Functions [23 marks]

Consider the following alternating series F_{ij} where $i, j \geq 1$:

$$
\begin{aligned}
F_{11}(n) &= f(n,1) - f(n,2) + f(n,3) - f(n,4) + f(n,5) - f(n,6) + f(n,7) - f(n,8) + \cdots f(n,n) \\
F_{21}(n) &= f(n,1) + f(n,2) - f(n,3) + f(n,4) + f(n,5) - f(n,6) + f(n,7) + f(n,8) + \cdots f(n,n) \\
F_{12}(n) &= f(n,1) - f(n,2) - f(n,3) + f(n,4) - f(n,5) - f(n,6) + f(n,7) - f(n,8) + \cdots f(n,n) \\
F_{31}(n) &= f(n,1) + f(n,2) + f(n,3) - f(n,4) + f(n,5) + f(n,6) + f(n,7) - f(n,8) + \cdots f(n,n)
\end{aligned}
$$

F_{ij} is similar to s_{ij} from Question 2(e). The only difference is that each term is now a function instead of a simple integer. Suppose the function `make_f(f,i,j)` will return a function that computes $F_{ij}(n)$. In other words, s_{11} from Question 2(a) can be expressed as:

```
s11 = make_f(lambda n,x: x, 1,1)
```

A. Consider the following alternating series to estimate $\ln 2$:

$$\ln 2 = 1 - 1/2 + 1/3 - 1/4 + \cdots + (-1)^{n+1} 1/n$$

and the function `ln2(n)` which gives the estimate to n terms as follows:

```
ln2 = make_f(<T1>, <T2>, <T3>)
```

Please provide possible implementations for the terms `T1`, `T2`, and `T3`. [5 marks]

`T1:` [3 marks]	**Full marks:** `lambda n,x:  1/x` **1 mark:** `lambda n,x:  <anything>` Something that resembles 1/x, provided the function arguments are correct.
`T2:` [1 marks]	`1`
`T3:` [1 marks]	`1`

B. Consider the following alternating series t_{ij}:

$$\begin{aligned}
t_{11}(n) &= n-(n-1)+(n-2)-(n-3)+(n-4)-(n-5)+(n-6)-(n-7)+\cdots 1 \\
t_{21}(n) &= n+(n-1)-(n-2)+(n-3)+(n-4)-(n-5)+(n-6)+(n-7)-\cdots 1 \\
t_{12}(n) &= n-(n-1)-(n-2)+(n-3)-(n-4)-(n-5)+(n-6)-(n-7)-\cdots 1 \\
t_{13}(n) &= n-(n-1)-(n-2)-(n-3)+(n-4)-(n-5)-(n-6)-(n-7)+\cdots 1
\end{aligned}$$

which is very similar to s_{ij} except the terms are reversed. Suppose the function `make_t` computes $t_{ij}(n)$ as follows:

```
def make_t(i,j):
    return make_f(<T4>, <T5>, <T6>)
```

Please provide possible implementations for the terms `T4`, `T5`, and `T6`. [5 marks]

`T4:`
[3 marks]

Full marks:
`lambda n,x:  n-x+1`

2 marks:
`lambda n,x:  n-x`

1 mark:
`lambda n,x:  <anything>`

`T5:`
[1 marks]

`i`

`T6:`
[1 marks]

`j`

C. Now suppose that the function `intsum(n)` will compute the sum of all positive integers from 1 up to and including *n*, and that:

```
intsum = make_f(<T7>, <T8>, <T9>)
```

Please provide possible implementations for the terms `T7`, `T8`, and `T9`. [5 marks]

Full marks:

```
T7:  lambda n,x:x if x%2 == 1 else -x
T8:  1
T9:  1
```

Full marks:

```
T7:  lambda n,x:x
T8:  n
T9:  1
```

4 marks: (Solution is almost correct, except for `T9`)

```
T7:  lambda n,x:x
T8:  n
```
`T9:` `0` (Not acceptable as $i, j \geq 1$)

3 marks: (Idea is correct, but `T9` is invalid. Thus not as correct as the above solution)

```
T7:  lambda n,x:x
T8:  1
```
`T9:` `0` (Not acceptable as $i, j \geq 1$)

D. It turns out that we can express `make_f` in terms of `fold` (see Appendix) as follows:

```
def make_f(f,i,j):
    def op(x,y):
        <T10>
    def f1(n,x):
        <T11>
    def helper(n):
        return fold(op,
                    lambda x: f1(n,x),
                    <T12>)
    return helper
```

Please provide possible implementations for the terms `T10`, `T11`, and `T12`. [8 marks]

`T10:` [2 marks]

```
return x+y
```

`T11:` [4 marks]

```
if x%(i+j)<i:
    return f(n,x+1)
else:
    return -f(n,x+1)
```

or

```
if x == 0:
       return 0
if x%(i+j) <= i and x%(i+j) > 0:
       return f(n,x)
else:
       return -f(n,x)
```

`T12:` [2 marks]

`n-1` or `n`, depending on the solution in `T11`.

`n` will be accepted if the answer in `T11` takes care of the base case where `x == 0`, even though the code might not be entirely correct.

Question 4: Espionage [23 marks]

Warning: Please read the question description clearly before you attempt this problem!!

You are working for a spy agency and you have been tasked with building a multi-level encryption system for the agency. The way this system works is that you create a message *m* with the function `make_message(msg)`, where `msg` is a String. A message `msg` is read with the `read(msg)` function.

After that you can encrypt your message *m* with the function `encrypt(m, password)` to return a new encrypted message m'. Any attempts to `read` the encrypted message m' will return the string `"*ENCRYPTED*"`. `decrypt(m', password)` will recover the message *m*. If the wrong password is used in the decryption, the returned message will be messed up and which will return the string `"*GARBLED*"` for any attempts to `read` it. `encrypt` can be applied multiple times and an equal number of `decrypt` with the passwords applied in strict reverse order will be required to recover the original message. Attempts to decrypt a non-encrypted message will have no effect.

Sample Execution:

```
>>> m1 = make_message("CS1010S is cool!")
>>> m1
<function make_message.<locals>.secret at 0x104074170>

>>> read(m1)
'CS1010S is cool!'

>>> m2 = encrypt(m1,12345)
>>> read(m2)
'*ENCRYPTED*'

>>> m3 = decrypt(m2,54321)
>>> read(m3)
'*GARBLED*'

>>> m4 = decrypt(m2,12345)
>>> read(m4)
'CS1010S is cool!'

>>> m5 = encrypt(m2,54321)
>>> read(m5)
'*ENCRYPTED*'

>>> m6 = decrypt(m5,54321)
>>> read(m6)
'*ENCRYPTED*'

>>> m7 = decrypt(m6,12345)
>>> read(m7)
'CS1010S is cool!'
```

```
>>> m8 = decrypt(m1,12345)
>>> read(m8)
'CS1010S is cool!'
```

A. Describe how you will keep track of the state of a message using a tuple. [3 marks]

A state of a message is a tuple including the original content and a set of encrypted passwords in the order that they are encrypted. For example, `('CS1010S is cool!', 12345, 54321)` is a tuple representing a state of a message whose original content is `'CS1010S is cool!'` and the message was encrypted with the password 12345 and then 54321.

Many students didn't seem to understand this question. It turns out that this is a standard question for data abstraction (ADT) questions. You first have to decide on what state you need to keep and also how to organize the state. In this case, the underlying data structure is tuple.

B. Write a function `make_message(msg)` that will take in a String representing the message and return a new message object that will be implemented according to your description in Part (A). Note that you cannot simply return an unobfuscated tuple because if not, then the secret message would be free for all to see. One simple approach to get around this is to wrap the object inside a function, but you are welcome to use another method to obfuscate the message. [4 marks]

```
def make_message(text):
    def secret():
       return (text,)
    return secret
```

This question tests 2 things: (i) whether the student understands how to create a new function and return it and (ii) whether the student can implement the data structure described in Part (A) correctly. -2 points for getting either wrong.

C. Please provide a possible implementation for the accessor function `read`. [4 marks]

```
def read(mes):
    stuff = mes()
    if len(stuff) == 1:
        return stuff[0]
    else:
        return "*ENCRYPTED*"
```

On average, we have generally been quite lenient in Parts (C) to (F). The failure to wrap the answers in a function to comply with Part (B) will result in -2 points for this whole question, i.e. you should not be penalized twice for the same mistake.

D. Please provide a possible implementation for the function `encrypt` that will encrypt the message in a manner consistent with your description in Part (A). [4 marks]

```
def encrypt(mes,password):
    def secret():
        return mes() + (password,)
    return secret
```

E. Please provide a possible implementation for the function `decrypt` that will decrypt the message that was encrypted by the `encrypt` function in Part (D) above. [4 marks]

```
def decrypt(mes,password):
    stuff = mes()
    if len(stuff) > 1:  # check if encrypted
       def secret():
          if password == stuff[-1]:
             return stuff[:-1]
          else:
             return ("*GARBLED*",)
       return secret
    else:
       return mes
```

Many students make the mistake of returning `"*GARBLED*"` directly. Note that `decrypt` needs to return an object (typically a function) and `"*GARBLED*"` is actually only returned in the `read` function.

F. Suppose now that we want the secret message to be decrypted by applying the required number of `decrypt` with the correct passwords *in any order*, though the decrypting of the message with any wrong password will garble the message. Explain how your implementation(s) in Parts (C) to (E) would need to be updated to implement this and write the required code. [4 marks]

We only need to change the `decrypt` function.

```
def decrypt(mes,password):

    stuff = mes()
    if len(stuff) > 1:  # check if encrypted
       def find(passwords, password):  # Returns remaining passwords
          if not passwords:
             return None
          elif password == passwords[0]:
             return passwords[1:]
          else:
             remaining = find(passwords[1:], password)
             if remaining != None:
                return (passwords[0],) + remaining
             else:
                return None

       remaining_passwords = find(stuff[1:], password)
       if remaining_passwords != None:
          def secret():
             return (stuff[0],) + remaining_passwords
          return secret
       else:
          return lambda :  ("*GARBLED*",)
    else:
       return mes
```

It is possible to have a shorter solution for this question. We share the above solution in the hope that it would be most straightforward and easy to understand.

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 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 —