CS1010S — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

28 September 2016 **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 **FOUR (4) questions** and **TWENTY-TWO (22) 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		
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 and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why. You may show your workings **outside the answer box** in the space beside the code. Partial marks will be awarded for workings if the final answer is wrong.

A. [5 marks]

```
x = 3
y = 5
def f(x):
    return x * y
def g(y):
    x = 7
    return f(x - y)
print(g(x))
```

```
20
```

B. [5 marks]

```
s = "Python is easy"
while not s == "":
   s = s[1:]
   out = s[0]
   if s == " ":
       break
print(out)
```

```
IndexError: string index out of range
```

C. [5 marks]

```
def foo(tup):
  if len(tup) <= 1:
    return (tup,)
  else:
    return (tup[0], tup[-1]) + foo(tup[1:-1])
print(foo(tuple(range(6))))
```

```
(0, 5, 1, 4, 2, 3, ())
```

D. [5 marks]

```
result = 1
for i in range(-6, 6, 2):
    if i % 3 == 0:
        result *= i
    elif i == 0:
        break
    else:
        result += i
print(result)
```

```
6
```

E. [5 marks]

```
x = 5
y = 10
z = 20
if x < z:
    y = y + x
if z < y:
    x = y + z
else:
    z = y - x
print(x, y, z)
```

```
5 15 10
```

F. [5 marks]

```
f = lambda x, y: y(y(x))
g = lambda x: lambda y: x(y)
print(f(lambda x:x+1, g)(4))
```

```
5
```

Question 2: Hangman [25 marks]

INSTRUCTION: You are not allowed to use any higher-order functions or any in-built Python string functions, e.g., `str.replace`, `str.count`, etc. for this question. String slicing is allowed as it is not a function.

In a game of Hangman, the player tries to uncover a hidden word by guessing the letters in the word. For this question, assume all words and letters are lowercase.

A. [Warm up] Write a function `count` which takes as input two strings, a letter and a word, and returns the number of occurrences of letter in the word.

Example:

```
>>> count('a', 'apple')
1

>>> count('z', 'apple')
0

>>> count('i', 'mississippi')
4
```

[4 marks]

```
def count(letter, word):
    result = 0
    for w in word:
        if w == letter:
            result += 1
    return result
```

or

```
def count(letter, word):
    if word == "":
        return 0
    elif letter == word[0]:
        return 1 + count(letter, word[1:])
    else:
        return count(letter, word[1:])
```

B. Is the function you wrote in Part (A) recursive or iterative? State the order of growth in terms of time and space for the function you wrote in Part (A). Explain your answer. [3 marks]

The function is **RECURSIVE / ITERATIVE** (circle one)

Time:
Iterative: $O(n)$, where n is the length of the input string.
Recursive: $O(n^2)$, where n is the length of the input string.

Space:
Iterative: $O(1)$, where n is the length of the input string.
Recursive: $O(n^2)$, where n is the length of the input string.

We mark according to the code written in part A, even though it might be wrong.

The function `remove` takes as inputs two strings, a letter and a word, and outputs the word with all occurrences of letter removed.

Example:

```
>>> remove('a', 'apple')
'pple'

>>> remove('p', 'apple')
'ale'
```

C. Provide an **iterative** implementation of the function `remove`. [4 marks]

```
def remove(letter, word):
    new_word = ""
    for i in word:
        if letter != i:
            new_word += i
    return new_word
```

D. What is the order of growth in terms of time and space for the function you wrote in Part (C). Briefly explain your answer.

[2 marks]

Time: $O(n^2)$ where n is the length on the input string. This is because in every loop iteration, the string concatenation of `new_word` gets longer until it is at worst, length n.

Space: $O(n)$, because even though there are no delayed operations or new objects being created every iteration, the new string created at worst can be the same length as the original string if there are no letters removed.

Other answers are acceptable according to what was written in part C. If the code does not make sense, then we cannot compute an order of growth, so no marks will be given.

E. Provide a recursive implementation of the function `remove`. [4 marks]

```
def remove(letter, word):
    if word == "":
        return ""
    elif letter == word[0]:
        return remove(letter, word[1:])
    else:
        return word[0] + remove(letter, word[1:])
```

F. What is the order of growth in terms of time and space for the function you wrote in Part (E). Briefly explain your answer.

Hint: Note the time and space complexity of string concatenation. [2 marks]

Time: $O(n^2)$ where n is the length of the string. This is because of string slicing at every of the n recursions.

Space: $O(n^2)$ where n is the length of the string. This is because of string slicing at every of the n recursions takes up space in the heap.

Other answers are acceptable according to what was written in part E. If the code does not make sense, then we cannot compute an order of growth, so no marks will be given.

Our Hangman game is played a little differently. The player starts with a score of 10 points and begins guessing with a letter. If the letter appears in the word, he will get 1 point added to his score for each occurrence of the letter in the word. e.g., if the word is "mississippi" and the player guesses "i", he will be awarded 4 points because there are 4 "i"s in the word.

Otherwise, if the player guesses a letter that is not in the word, 1 point will be deducted from his score.

The player continues guessing and the game ends once the entire word is revealed by having all it letters guessed.

G. The function `hangman` takes as inputs two strings, the word and a sequence of guesses. The guesses are the letter to be guessed in sequence by the player. The game is played using this sequence of guesses and the function outputs the final score of the game. You can assume that the sequence of guesses will include all the letters of the word.

Example:

```
>>> hangman("apple", "abcdefghijklmnop")
3

>>> hangman("apple", "alpebqrs")
15

>>> hangman("apple", "zyxwvutsrqponmlkjihgfedcba")
-7
```

Provide an implementation of the function `hangman`.

Hint: You may use the functions `remove` and `count` which you have defined previously.

[6 marks]

```
def hangman(word, guess):
    score = 10
    for letter in guess:
        c = count(letter, word)
        if c > 0:
            score += count(letter, word)
            word = remove(letter, word)
        else:
            score = score - 1
        if word == "":
            break
    return score
```

Question 3: Higher-Order Hangman [23 marks]

INSTRUCTION: Parts A, B and C should be solved using the given higher-order function, and not by recursion, iteration or reusing the functions defined in Question 2.

A. It turns out that the function `count` in Question 2A can be defined in terms of the higher-order function `sum` (refer to the Appendix) as follows:

```
def count(letter, word):
    <PRE>
    return sum(<T1>,
               <T2>,
               <T3>,
               <T4>)
```

Please provide possible implementations for the terms `T1`, `T2`, `T3` and `T4`. You may also optionally define functions in `<PRE>` if needed. [5 marks]

Correct answer, which uses `sum` to increment an index:

```
def count(letter, word):
    return sum(lambda x: 1 if word[x] == letter else 0,
               0,
               lambda x: x+1,
               len(word)-1)
```

Note the ending condition is `len(word)-1` due of the implementation of `sum` (terminates when `a > b`).

Also note the use of the ternary form of `if-else` statement. This form is needed when using `if-else` in the body of `lambda` because it is a one-line expression, not a "block structure". It is ok not to use this form and define a new function using the block structure in the `<PRE>` section like so:

```
def count(letter, word):
    def term(x):
        if word[x] == letter:
            return 1
        else:
            return 0

    return sum(term,
               0,
               lambda x: x+1,
               len(word)-1)
```

This question is marked as a whole, so simply filling common values in the parts will not give you the marks. There has to be some understanding shown in the answer.

B. Caryn thinks that since the + operator can also be used to concatenate strings together, she can also use the higher-order function `sum` to implement the function `remove` (from Question 2C) by doing:

```
def remove(letter, word):
    return sum(...)
```

Do you think Caryn is correct? If yes, please provide a possible implementation. If no, please explain why and show what modifications to `sum` is needed to enable Caryn's idea to work.

[4 marks]

No, Caryn is not correct. Because the base value for the sum function is 0, and + will result in an error when used with a `str` and `int`.
She'll need to modify sum by returning empty string `''` as the base value, i.e.,

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

C. Now we can definitely define the function `remove` from Question 2C in terms of `fold` as follows:

```
def remove(letter, word):
    <PRE>
    return fold(<T5>,
                <T6>,
                <T7>)
```

Please provide possible implementations for the terms `T5`, `T6` and `T7`. To keep things simple, you can assume that `word` is not an empty string. You may also optionally define functions in `<PRE>` if needed. [5 marks]

```
def remove(letter, word):
    return fold(lambda a,b: b+a, # note the order!
                lambda x: "" if word[x] == letter else word[x],
                len(word)-1)
```

D. Using the `hangman` function to play the game is not fun because it needs to take in both the word and the guesses as an input, thus exposing the word to the player. Beni wants to create a hangman game using a function `create_hangman` that takes as input a word, and outputs a function that she can pass to her friends to play the game.

Example:

```
>>> game = create_hangman("hello")
>>> game("abcdefghijklmnop")
4

>>> game("oleh")
15

>>> game2 = create_hangman("pikachu")
>>> game2("abcdefghijklmnopqrstuvwxyz")
3
```

Provide possible implementations for the function `create_hangman`. You may reuse the functions defined in Question 2. [4 marks]

```
def create_hangman(word):
    return lambda guess: hangman(word, guess)
```

E. **[Warning: Very Challenging!]** Professor Siva thinks he can do better. Instead of having to input all the guesses in one string at a go, he believes he can use the power of higher-order functions to allow the game to accept one guess at a time and print the current score after each guess. He calls his function `ultimate_hangman` and it works as follows:

```
>>> game = ultimate_hangman("apple")
'Score is 10'

>>> game = game("a")
'Score is 11'

>>> game = game("p")
'Score is 13'
```

```
>>> game = game("o")
'Score is 12'

>>> game = game("l")
'Score is 13'

>>> game = game("e")
'Game over!'
'Score is 14'

>>> game = game("w")
'Game over!'
'Score is 14'
```

Unfortunately, Professor Siva is stuck coding halfway. Please help him complete his ambitious function by filling in the body of the inner function `guess`. [5 marks]

```
def ultimate_hangman(word):
    def helper(remain, score):
        def guess(letter):
            add_score = count(letter, remain)
            new_remain = remove(letter, remain)
            if new_remain == "":
                print("Game over!")
                return helper(new_remain, score+add_score)
            elif add_score == 0:
                return helper(remain, score-1)
            else:
                return helper(new_remain, score+add_score)
        print("Score is", score)
        return guess
    return helper(word, 10)
```

Question 4: Pokémon Eggs [22 marks]

Warning: Please read the entire question clearly before you attempt this problem!! You are also not to use any Python data types which have not yet been taught in class.

In the game Pokémon Go!, Pokémon can be hatched from eggs by incubating and walking them for a certain distance. Eggs are created with a certain distance to walk and the name of the Pokémon which will hatch from it.

The following functions support the egg data type:

- `create_egg(distance, name)` takes as input a distance and the name of the Pokémon that will eventually hatch from it and returns an egg. You will get 1 bonus mark if the Pokémon is not revealed by someone breaking abstraction and attempting to "print" the egg (See sample run below). You may use any method that works to hide it.
- `distance_left(egg)` takes as input an egg, and returns the remaining distance to walk in order to hatch the egg. Once an egg's remaining distance reaches 0, it will be considered to have hatched. The remaining distance can not be negative.
- `get_pokemon(egg)` takes as input an egg, and returns the name of the Pokémon in the egg if the egg has hatched. Otherwise it returns the string `"Walk some more!"`.
- `walk_egg(distance, egg)` takes as input a distance which is a float, and an egg. It returns an egg with the remaining distance to walk reduced by the distance given in the input.

Sample run:

```
>>> egg = create_egg(2, "Pidgey")
>>> print(egg)  # trying to cheat by printing egg contents
<function create_egg.<locals>.<lambda> at 0x00000224A77A5D90>

>>> distance_left(egg)
2
>>> egg = walk_egg(0.8, egg)
>>> get_pokemon(egg)
'Walk some more!'

>>> distance_left(egg)
1.2

>>> egg = walk_egg(0.8, egg)
>>> distance_left(egg)
0.3999999999999999  # remember float inaccuracies

>>> egg = walk_egg(0.8, egg)
>>> distance_left(egg)
0
>>> get_pokemon(egg)
'Pidgey'
```

A. Explain how you will use tuples to represent an egg. 1 bonus mark if you can explain how you plan to hide the Pokémon name from would be "cheaters". [2 marks]

An egg will just be a tuple of two elements, the first being the distance left to hatch the egg, and the second the name of the Pokémon. We will wrap the second element (the name) in a lambda function to obscure it.

B. Provide an implementation for the functions `create_egg`, `walk_egg`, `get_pokemon` and `distance_left`. 1 bonus mark if you implement the hiding of the Pokémon name in an egg. [6 marks]

```
def create_egg(distance, pokemon):
    return (distance, lambda: pokemon)



def get_pokemon(egg):
        if egg[0] <= 0:
        return egg[1]()
    else:
        return "Walk some more!"



def distance_left(egg):
    return egg[0]
```

```
def walk_egg(distance, egg):
    new_distance = max(0, egg[0] - distance)
    return (new_distance, egg[1])
```

[Important!] For the remaining parts of this question, **you should not break the abstraction of an egg.**

Pokémon trainers carry their Pokémons and eggs in a bag. You are to design and implement a data structure to represent a bag that supports the following functions:

- `empty_bag()` takes no inputs and return an empty bag.
- `list_pokemons(bag)` takes as input a bag, and returns a tuple of the names of Pokémons contained in the bag.
- `list_eggs(bag)` takes as input a bag, and returns a tuple of distance remaining to hatch each unhatched egg in the bag.
- `put_egg(bag, egg)` takes as input a bag and an egg, and returns a new bag with the egg added into it.

C. Explain how you will use tuples to represent a bag. [2 marks]

You can implement a bag however you want, as long as it makes sense. Here are two common methods:

1. Use a tuple containing two tuples, which contains Pokémons and eggs respectively.
2. Use a tuple where the elements are simply eggs. Pokémons do not have to be actual objects anyway so their names can be extracted from the hatched eggs.

D. Provide an implementation for the functions `empty_bag`, `list_pokemons`, `list_eggs` and `put_egg`. [6 marks]

Functions for using method 1 representation.

```
def empty_bag():
    return ((), ())

def put_egg(bag, egg):
    return (bag[0], bag[1] + (egg,))

def list_pokemons(bag):
    return bag[0]

def list_eggs(bag):
    eggs = ()
    for egg in bag[1]:
        eggs = eggs + (distance_left(egg),)
    return eggs
```

Functions for method 2 representation.

```
def empty_bag():
    return ()

def put_egg(bag, egg):
    return bag + (egg,)

def list_pokemons(bag):
    return tuple((get_pokemon(x) for x in bag if distance_left(x) == 0))

def list_eggs(bag):
    return tuple((distance_left(x) for x in bag if distance_left(x) > 0))
```

The functions shown here uses Python comprehension which was taught in Lecture 7. You can, of course, always use `for`-loops or `map` & `filter`. We just want to showcase the "power" and simplicity of comprehension here.

To hatch the eggs, a Pokémon trainer must walk the distance required by the eggs. Once the remaining distance of an egg reaches 0, it will then hatch. The hatched Pokémon should automatically be added to the bag while the hatched egg will be removed.

The function `walk` takes as input the distance walked and a bag, and outputs a bag which reflects the new state after walking for the distance.

Example:

```
>>> bag = empty_bag()
>>> bag = put_egg(bag, create_egg(2, "Pidgey"))
>>> bag = put_egg(bag, create_egg(5, "Horsea"))
>>> list_pokemons(bag)
()
>>> list_eggs(bag)
(2, 5)

>>> bag = walk(3, bag)  # Pidgey egg has hatched
>>> list_eggs(bag)
(2,)
>>> list_pokemons(bag)
('Pidgey',)

>>> bag = walk(3, bag)  # Horsea egg has hatched
>>> list_eggs(bag)
()
>>> list_pokemons(bag)
('Pidgey', 'Horsea')
```

E. Provide an implementation of the function `walk`. [6 marks]

Using method 1 of representation

```
def walk(distance, bag):
    pokemon_bag = bag[0]
    egg_bag = ()
    for egg in bag[1]:
        new_egg = walk_egg(distance, egg)
        if distance_left(new_egg) == 0:
            pokemon_bag = pokemon_bag + (get_pokemon(new_egg),)
        else:
            egg_bag = egg_bag + (new_egg,)
    return (pokemon_bag, egg_bag)
```

Using method 2 of representation

```
def walk(distance, bag):
    new_bag = tuple((walk_egg(distance, egg) for egg in bag))
```

See how simple it becomes with a different representation. =)

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 map(fn, seq):
    if seq == ():
        return ()
    else:
        return (fn(seq[0]),) + map(fn, seq[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

Scratch Paper

Scratch Paper

— E N D O F P A P E R —