CS1010S — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

18 March 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 **EIGHTEEN (18) 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 pages marked "scratch paper" in the question set may be used as scratch paper.
8. You are allowed to detach 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 any other writing instrument as you like, as long as it is not red in colour.

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]

```python
r = 0
for i in range(10):
    r += i // 3
    if i % 2:
        i += 2
print(r)
```

```
12
```

B. [5 marks]

```python
def f(n):
    if n == 0:
        return False
    elif n == 1:
        return True
    else:
        return f(n/2)
print(f(48))
```

Infinite loop. RecursionError.

C. [5 marks]

```python
def foo(x):
    return lambda y: x
def bar(x):
    return lambda x: x
def baz(x):
    return lambda y: y(x)
print(foo(bar)(baz)(foo)(baz)(bar)(foo)(1)(2)(3))
```

```
3
```

D. [5 marks]

```
a, b = 4, 10
if a > b:
    b = a - b
    if b > a:
        b = b - a
else:
    a = b + a
    if b > a:
        a = b - a
if a > b:
    b = a - b
print(a)
print(b)
```

```
14
4
```

E. [5 marks]

```
tup = (4, 'a', 3, 'b', 2, 'c', 1, 'd')
while tup:
    if type(tup[0]) != int:
        break
    if len(tup) < tup[0]:
        continue
    tup = tup[tup[0]:]
print(tup)
```

```
('d',)
```

F. [5 marks]

```
x, y, z = 7, 2, 1
def boo(x):
    y = x + 2
    z = 3
    return bob(y)
def bob(x):
    return x + y + z
print(boo(z))
```

```
6
```

Question 2: Golf [19 marks]

"Golf is a club and ball sport in which players use various clubs to hit balls into a series of holes on a course in as few strokes as possible." (Wikipedia)

A game of golf is scored based on the number of strokes taken to hit the ball from the tee (starting point) into the hole (the goal). Thus, a game of golf use be represented as a tuple of integers where each integer is the number of strokes a player took for each hole. The first element represents the strokes taken for the first hole, the second for the second hole and so on.

A. **[Warm up]** Without using any higher-order functions, write a function `total_strokes` that takes as input a game of golf and returns the total number strokes the player took for the entire game. [4 marks]

```
def total_strokes(game):
    total = 0
    for i in game:
        total += i
    return total
```

or

```
def total_strokes(game):
    if game:
        return game[0] + total_strokes(game[1:])
    else:
        return 0
```

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 `game`.
Recursive: $O(n^2)$, where n is the length of `game`.

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

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

Because there is no standardized field for golf, each hole has its own unique terrain and features. Thus, each hole is given a par value, which is the number of strokes a skilled golfer should be required to complete the hole.

The score for each hole is then the difference between the number of strokes the player took and the par for that hole. For example, if the par of the hole is 4 and the player took only 2 strokes to complete, his score for the hole will be -2. If he took 5 strokes to complete, his score will be 1. The total score for a game is simply the sum of the scores of all the holes.

The function `par` takes as input an integer representing the current hole, and returns the par value for the hole. For example, if the first hole has a par of 4, then `par(0)` will return `4`. You may assume this function is given.

C. Write a **recursive** function `compute_score(game)` which takes in a game and returns the score of the game. [4 marks]

```
def compute_score(game):
    def helper(i):
        if i == len(game):
            return 0
        else:
            return game[i] - par(i) + helper(i+1)
    return helper(0)
```

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: $O(n)$ where n is the length of game. This is because helper is called n times.

Space: $O(n)$ where n is the length of the string. This is because there are n recursions, taking up space on the stack.

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. Write an iterative function `compute_score(game)` which takes in a game and returns the score of the game. [4 marks]

```
def compute_score(game):
    score = 0
    for i in range(len(game)):
        score += game(i) - par(i)
    return score
```

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. [2 marks]

Time: $O(n)$ where n is the length of game. This is because the for-loop loops n times.

Space: $O(1)$, because there are no delayed operations or new objects being created every iteration.

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.

Question 3: Higher-Order Function [23 marks]

For this question, you are not to reuse the functions defined in Question 2.

A. [Warm-up] The function `compute_score` from Question 2C can be expressed in terms of the higher-order function `sum` as follows:

```
def compute_score(game):
    <PRE>
    return sum(<T1>,
               <T2>,
               <T3>,
               <T4>)
```

Please provide possible implementations for the terms `T1`, `T2`, `T3` and `T4`. You may assume the function `par` described in Question 2 is available. You may also optionally define functions in `<PRE>` if needed. [5 marks]

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

```
def compute_score(game):
    return sum(lambda x: game[x] - par(x)),
               0,
               lambda x: x+1,
               len(game)-1)
```

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

B. The function `game_to_scores` takes as input a golf game of strokes (as described in Q1), and returns a tuple of scores for each hole. Recall that the score for a hole is the difference between the number of strokes taken and the par for the hole.

For example, if the par for all the holes is 3, then `game_to_scores((4, 3, 2, 4, 3))` will return `(1, 0, -1, 1, 0)`.

Provide an implementation of `game_to_scores`, by **using only the higher-order functions** listed in the Appendix. Half the marks will be given if the solution involves recursion or iteration. [3 marks]

```
def game_to_scores(game):
    return map(lambda x: game[x] - par(x),
               enumerate_interval(0, len(game)-1))
```

C. In golf, common scores for a hole have specific terms. For example, a score of +1 is called Bogey and a score of -1 is called a Birdie.

The function `num_score` takes as inputs a game and a score, and returns the number of holes which has the given score.

For example, if the par for all the holes is 3, `num_score((2, 3, 2, 4, 3), -1)` will return 2.

Provide an implementation of `num_score`, by **using only the higher-order functions** listed in the Appendix. You may reuse the function `game_to_scores` defined in the previous part. Half the marks will be given if the solution involves recursion or iteration. [3 marks]

```
def num_score(game, n):
    return len(filter(lambda x:x == n, game_to_scores(game)))
```

D. **[Challenging]** We can solve 3C using a technique called MapReduce, which first applies the higher-order function `map` followed by `accumulate` to a sequence of data as follows:

```
def num_score(game, n):
    <PRE>
    return accumulate(<T5>,
                      <T6>,
                      map(<T7>,
                          <T8>))
```

Please provide possible implementations for the terms `T5`, `T6`, `T7` and `T8`. You may also optionally define functions in `<PRE>` if needed. [6 marks]

*optional `<PRE>`:	
`<T5>`:	`lambda x, y: x+y`
`<T6>`:	`0`
`<T7>`:	`lambda x: 1 if x == n else 0`
`<T8>`:	`game_to_score(game)`

E. We can define specific functions to count the number of occurrence of a particular score in a game. For example, a function `count_birdie(game)` will return the number of birdies in the game.

Such functions can be defined using a function `make_counter`, that takes as input the score to be counted.

Example, assuming par for all the holes is 3:

```
>>> count_birdie = make_counter(-1)
>>> count_birdie((2, 3, 2, 4, 3))
2
```

Provide an implementation of `make_counter`. You may use functions previously defined in this question. [3 marks]

```
def make_counter(n):
    return lambda game: num_score(game, n)
```

F. So far, we have been using wishful thinking by assuming that the function `par` is provided. However, different golf courses would have different pars for their holes.

The function `set_par` takes as input a tuple of integers that represent the par for each of the holes of a golf course, and returns the function `par`.

Example:

```
>>> par = set_par((3, 3, 5, 4, 3))
>>> game_to_score((4, 3, 5, 3, 5))
(1, 0, 0, -1, 2)
```

Provide an implementation of `set_par`. [3 marks]

```
def set_par(pars):
    return lambda x: pars[x]
```

Question 4: Golf Course [28 marks]

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

A golf course would like to keep a record of the scores of all the players who played on the course. You are tasked to propose a way to do implement this scorecard in Python. Note that although the players are anonymous, you should be able to keep track of the scores for the individual holes.

A typical golf course has 18 holes, but your scorecard should be able to handle courses of different number of holes. You will have to support and implement the following functions:

- `create_new_scorecard(n)` takes as input the number of holes of the course and returns an empty scorecard.
- `num_holes(scorecard)` takes as input a scorecard for a course, and returns the number of holes of the course.
- `add_to_scorecard(scorecard, score)` takes two inputs, a scorecard and tuple containing the score (not stroke) for each hole played on the course and returns a scorecard with the player's scores added.
- `get_scores(hole)` takes as input a hole (an integer starting from 1) and returns a tuple of all the scores attained for that hole.

A. State and describe the data structure you will use and to implement a scorecard. You may use lists if you wish. [2 marks]

There are two main ways to implement scorecard.

1. Use a tuple/list with the first element be the number of holes, and the second element be a list of scores.
2. Use a tuple/list of *n* elements, where each element is a list of

B. Provide an implementation for the functions `create_new_scorecard`, `num_holes` and `get_scores`. [4 marks]

```python
def create_new_scorecard(n):
    return(n, ())

def create_new_scorecard(n):
    return ((),) * n
```

```
def num_holes(scorecard):
    return scorecard[0]

def num_holes(scorecard):
    return len(scorecard)


def get_scores(scorecard, hole):
    return map(lambda x:x[hole-1], scorecard[1])

def get_scores(scorecard, hole):
    return scorecard[n-1]
```

C. The function `combine_scorecards` takes as input two scorecards, and return a scorecard with the combined scores.

Provide an implementation for the functions `add_to_scorecard` and `combine_scorecards`. Assume that the length of the score matches the number of holes of the scorecard and that the inputs are correct. [6 marks]

```
def add_to_scorecard(scorecard, score):
    return (scorecard[0], scorecard[1] + (score,))

def add_to_scorecard(scorecard, score):
    new_scorecard = ()
    for i in range(len(scorecard)):
        new_scorecard += (scorecard[i] + (scores[i],),)
    return new_scorecard



def combine_scorecards(sc1, sc2):
        return (sc1[0], sc1 + sc2)

def combine_scorecards(sc1, sc2):
    new_scorecard = ()
    for i in range(len(sc1)):
        new_scorecard += (sc1[i] + sc2[i],)
    return new_scorecard
```

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

D. Write a function `average_score` which takes as inputs a scorecard and a hole, and returns the average score for the hole. You may use the Python built-in `sum` function instead of the one found in the Appendix. [4 marks]

```
def average_score(scorecard, hole):
    scores = get_scores(scorecard, hole)
        return sum(scores)/len(scores)
```

-2 marks for breaking abstraction, since it is stated explicitly above!

E. The course managers wish to know the number of birdies or bogeys or other scores that were obtained on the course.

Write a function `count_score` that takes as input a scorecard and a score, and returns the number of occurrence of score across all holes in the scorecard. [4 marks]

```
def count_score(scorecard, score):
    total = 0
    for i in range(1, num_hole(scorecard)+1):
        total += len(filter(lambda x:x == score, get_scores(scorecard, i)))
```

F. Oh no! Apparently somebody has been using the wrong function to combine scorecards together. He has been using `add_to_scorecard` instead of `combine_scorecards`.

According to your scorecard implementation, explain what happens and describe the result of using `add_to_scorecard` with two scorecards as the inputs, assuming the function does not validate the inputs. [4 marks]

In both implementation, the score will be wrongly appended to the tuple of scores, resulting in some of the elements as integers instead of a tuple.

G. One way to prevent Part F from happening is to check if the input is a score or a scorecard. Implement a function `is_score(score)` that returns `True` if the input is indeed a score, and `False` if it is a scorecard. [4 marks]

```
def is_score(score):
    for i in score:
        if type(i) == tuple:
            return False
    return True
```

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