CS1010FC — Programming Methodology
School of Computing
National University of Singapore

Solutions for Mid-Term Test

28 March 2015 **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 test**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FIVE (5) questions** and **SIXTEEN (16) 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 [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.
```
y = 97%4
if y%3:
    print("Hello")
else:
    print("There!")
```

[5 marks]

```
Hello
```

This is to test that students understand `if-else` and also the modulo operator (`%`).

B.
```
x = 123
y = 1000
def foo(x,y):
    y = 100
    print(x//y)
foo(y,x)
```

[5 marks]

```
10
```

This is to test that students understand simple function calls, scope and also the quotient (`//`) operator.

C.
```
def f1(x,y,z):
    return x(y,z)
def f2(x,y):
    return y(x)
print(f1(lambda x,y: x+1, 2, lambda x: x**2))
```

[5 marks]

```
3
```

This is to test that students understand that `lambda` creates a function that can be passed as an argument into a function.

D.
```
def count():
    for i in range(10):
        total = i
        for j in range(10):
            total += j
            if j%2==0:
                continue
            elif j+i>15:
                return total
print(count())
```

[5 marks]

```
52
```

This is to test that students understand `for` loops and also `continue` and `break`.

E.
```
x = ("a","b","c")
y = ("a")
z = (x, ("a", "c"), ("a","b"), ("b","c"))
if x in z:
    if y in x:
        print("Good")
    elif y in z:
        print("Bad")
    print("Done")
```

[5 marks]

```
Good
Done
```

This is to test that students understand the `in` operator for tuples and also what happens when there is no `else`. 3 points is taken off for just `"Good"`.

F.
```
def p():
    return lambda: print("Confused?")
if not p:
    p()
elif p:
    p()()
else:
    print(not p()())
```

[5 marks]

```
Confused?
```

This is to test that students understand what happens when functions have no arguments and that objects that are not `False` will be considered `True`.

Question 2: Game of Mystery Boxes [20 marks]

In the Game of Mystery Boxes, contestants are given a sequence of boxes. Each box contains either a positive or negative cash amount. For every box the contestant opens, the amount of cash in the box will be added (or subtracted if the amount is negative) to the contestant's winnings. The contestant can decide at any time whether to continue to open the next box, or walk away with the current winnings. The contestant starts with $0 and the game ends if the current winnings goes below 0 and the contestant will leave the game with $0 in winnings.

A. Write a function `winnings` that takes as input a tuple of integers which represent the cash amount of the sequence of boxes that a contestant can open and returns the final winnings if all the boxes were to be opened (until the game ends). You may assume the contestant chooses to open boxes until there are no more boxes left, or he loses with $0. [6 marks]

```
def winnings(seq):
    total = 0
    for i in seq:
        total += i
        if total < 0:
            return 0
    return total
```

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 where n is the length of `seq`. Explain your answer. [4 marks]

Time: $O(n)$, since the loop will run n times.

Space: $O(1)$, only 2 variables `total` and `i`.

C. Hindsight is 20/20. Write a function `max_winnings(seq)` that takes as input a tuple of integers `seq` which represent the cash amount of the boxes in sequence, and return the maximum possible winnings, i.e. we assume that the contestant will decide whether to open the boxes based on the optimal strategy. [6 marks]

```
def max_winnings(seq):
    total = 0
    max = 0
    for i in seq:
        total += i
        if total > max:
            max = total
        if total < 0:
            return max
    return max
```

This question is meant to test the students' familiarity with loops.

D. Now suppose instead there are an infinite number of boxes, and that there is a function `next_box()` that returns the cash value of the next box that is opened. Assume that the contestant will keep opening boxes until his winnings fall below zero and the game ends. Write a function `max_num` that returns the number of boxes that a contestant can open before the game ends. [4 marks]

```
def max_num():
    total = 0
    steps = 0
    while total >= 0:
        total += next_box()
        steps += 1
    return steps
```

Question 3: Higher-Order Functions [24 marks]

Consider the following higher-order function that we call `choosy`:

```
def choosy(select, a, b, op, n, base):
    if n == 1:
        return base
    elif select(n):
        return op(a(n), choosy(select, a, b, op, n-1, base))
    else:
        return op(b(n), choosy(select, a, b, op, n-1, base))
```

A. [Warm-up] Suppose the function `sum_integers(n)` computes the sum of integers from 1 to *n* (inclusive) and `sum_integers(n)` is defined as follows:

```
def sum_integers(n):
    return choosy(<T1>,
                  <T2>,
                  <T3>,
                  <T4>,
                  <T5>,
                  <T6>)
```

Please provide possible implementations for T1, T2, T3, T4, T5 and T6. [8 marks]

`<T1>`: [2 marks]	`lambda x: True`
`<T2>`: [1 mark]	`lambda x: x`
`<T3>`: [1 mark]	`lambda x: x`
`<T4>`: [2 marks]	`lambda x, y: x+y`
`<T5>`: [1 mark]	`n`
`<T6>`: [1 mark]	`1`

B. Suppose the function `sum_even_integers(n)` computes the sum of even integers from 1 to *n* (inclusive) and `sum_even_integers(n)` is defined as follows:

```
def sum_even_integers(n):
    return choosy(<T7>,
                  <T8>,
                  <T9>,
                  <T10>,
                  <T11>,
                  <T12>)
```

Please provide possible implementations for T7, T8, T9, T10, T11 and T12. [8 marks]

`<T7>:` [2 marks]	`lambda x: x % 2 == 0`
`<T8>:` [1 mark]	`lambda x: x`
`<T9>:` [1 mark]	`lambda x: 0`
`<T10>:` [2 marks]	`lambda x, y: x+y`
`<T11>:` [1 mark]	`n`
`<T12>:` [1 mark]	`0`

C. Suppose the function `is_prime(n)` returns `True` is the number n is prime, for $n \geq 2$, and `False` otherwise, and `is_prime` is defined as follows:

```
def is_prime(n):
    def helper(x):
        return choosy(<T13>,
                      <T14>,
                      <T15>,
                      <T16>,
                      <T17>,
                      <T18>)
    return helper(n-1)
```

Please provide possible implementations for T13, T14, T15, T16, T17 and T18. [8 marks]

Hint: The boolean operator `and` will return `True` if and only if both inputs are `True`, and `False` otherwise.

`<T13>`: [2 marks]	`lambda x: n % x == 0`
`<T14>`: [1 mark]	`lambda x: False`
`<T15>`: [1 mark]	`lambda x: True`
`<T16>`: [2 marks]	`lambda x, y: x and y`
`<T17>`: [1 mark]	`x`
`<T18>`: [1 mark]	`True`

Question 4: Tic-Tac-Toe [23 marks]

We are all familiar with this childhood game called Tic-Tac-Toe, which is played on a 3×3 grid and two players take turns to put in either a symbol "X" or "O" in one of the squares. The first player to place 3 of his symbols in a row either vertically, horizontally, or diagonally, wins. In this problem, you will model this game.

The following is a brief description of the problem:

- `make_state()` will create an initially empty state object for the game.
- `get_square(s,x,y)` will return the symbol in the game state object `s` at the grid location (x,y), where $0 \leq x,y \leq 2$. If the grid location is empty, `get_square(s,x,y)` will return `None`.
- `move(s,x,y,player)` will return a new state object after a player (where player is either `"X"` or `"O"` makes a move at the grid location (x,y) for a game state object `s`. If the move is invalid, `False` is returned instead.

A. Decide on an implementation for the Tic-Tac-Toe game state object and implement `make_state`. Describe how the state is stored in your implementation and the conditions that a valid game state object in your implementation must satisfy. [4 marks]

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

```
def make_state():
    return ((0,0,0),(0,0,0),(0,0,0))
```

The game state is represented as a tuples of 3 tuples, each with 3 elements. We use 0 to represent an empty square, 1 to present `"X"` and 2 to represent `"O"`.

The key lesson in this problem is that it is important to choose a good representation. -1 points if symbols are not explained properly.

B. Implement a function `check_rep(s)`, that will return `True` is `s` a valid game state object for the representation you described in Part(A) above, or `False` otherwise. You can assume that `s` is a tuple. [4 marks]

```
def check_rep(s):
    if len(s) != 3: # Check there are 3 rows
        return False
    for row in s: # Check there are 3 columns
        if len(row) != 3:
            return False
    for row in s:
        for col in row:
            if col not in (0,1,2):  # Check that all elements are valid
                return False
    return True
```

-2 points for failure to check on symbols. Basically, need to check that size/format is correct and that all the symbols are valid.

C. Implement a function `get_square(s,x,y)`, that will return the symbol in the square at grid position $(x,y), 0 \leq x,y \leq 2$ for the game state object `s`. If the square is empty, return `None`. [3 marks]

```
def get_square(s,x,y):
    if s[x][y] == 1:
        return "X"
    elif s[x][y] == 2:
        return "O"
    elif s[x][y] == 0:
        return None
```

D. Implement a function `move(s,x,y,player)`, that will return a new state object after a player (where player is either `"X"` or `"O"`) makes a move at the grid location (x,y) for a game state object `s`. If the move is invalid, `False` is returned instead. [6 marks]

```
def move(s,x,y,player):
    if s[x][y] != 0:  # Make sure that box is empty
        return False
    elif player not in ("X","O"): # Make sure that player is valid
        return False

    symbol = 1        # Determine the symbol for the new entry
    if player == "O":
        symbol = 2

    result = ()       # Construct and return new state
    for row in range(3):
        temp = ()
        for col in range(3):
            if row==x and col==y:
                temp += (symbol,)
            else:
                temp += (s[row][col],)
        result += (temp,)
    return result
```

E. Implement a function `check_win(s)`, that will return the symbol for the winning player (where player is either `"X"` or `"O"` for a game state object `s`, or `False` if neither player has won. [6 marks]

```
def check_win(s):
    # Check diagonals:
    win = s[0][0]*s[1][1]*s[2][2]
    if win not in (1,8):
        win = s[0][2]*s[1][1]*s[2][0]

    # Check rows:
    for row in s:
        if win not in (1,8):
            win = row[0]*row[1]*row[2]

    # Check cols:
    for col in range(3):
        if win not in (1,8):
            win = s[0][col]*s[1][col]*s[2][col]

    if win == 1:
        return "X"
    elif win == 8:
        return "O"
    else:
        return False
```

Brute force is acceptable. There are 8 cases: 3 vertical, 3 horizontal and 2 diagonal. -1 point for failure to check that the squares are not empty, i.e. some students just check that 3 squares are the same and not that they are not empty.

Question 5: What is the purpose of education to you? [3 marks]

> *Finally, the end product [of education] is the good citizen, the man or woman who has had the maximum of nurturing of his or her natural talent to fit him or her to earn his or her livelihood in our society and who can bring up a family and care for them.* *– Mr Lee Kuan Yew*

Do you agree? Explain. (Basically, tell us: why are you coming to university?)

Student will get points for any reasonably well-articulated answer.

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

— END OF PAPER —