NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1, 2013/2014

CS1010S - PROGRAMMING METHODOLOGY

27 November 2013 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. The examination paper contains **SIX (6) questions** and comprises **TWENTY-TWO (22) pages**.
2. Weightage of questions is given in square brackets. The maximum attainable score is 100.
3. This is a **CLOSED** book examination, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
4. Write all your answers in the space provided in this booklet.
5. **Please write your matriculation number below.**

MATRICULATION NUMBER: ______________________________

(this portion is for the examiner's use only)

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Q6		
Total		

Question 1: Warm Up [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. If the interpreter produces an error message, or enters an infinite loop, explain why.

A. [4 marks]

```
a = [[1],[2],[3]]
b = a.copy()
a[0] = b[2]
print(a if a[1] is b[1] else b[1])
```

B. [4 marks]

```
try:
    y = 21
    x = y%4
    z = y%5
    while x < y:
        y = y//(z-x)
except ZeroDivisionError:
    print("Whoops")
except:
    print("Some other error")
else:
    print("Cool!")
```

C. [4 marks]

```
a = [1,4,3,2]
def fuddle(lst):
    temp = list(lst)
    temp.sort()
    temp.append(32)
    lst = temp
fuddle(a)
print(a)
```

D. [4 marks]

```
def santa(*says):
    if not says:
        return "ho! xmas!"
    else:
        return says[0]+"! "+santa(*says[1:])
print(santa("ho","ho","ho"))
```

E. [4 marks]

```
a = 0
b = 10
while a < b:
    a = b % 3 + 1
    b = b - a
print(b)
```

F. [4 marks]

```
def a(x,*arg):
    result = x
    for f in arg:
        result = f(result)
    return result
def b(x):
    return 2*x
def c(x):
    return x+1
print(a(3,b,c,c,b))
```

Question 2: List Processing [20 marks]

Suppose you are given the following lists:

```
a = [1,2,3,4,5,6,7,8,9,10]
b = [10,9,8,7,6,5,4,3,2,1]
c = [1,0,1,0,1,0,2,0,2,0]
d = [5,1,1,1,5]
e = [-1,1,-2,2,-1,1]
```

A. Give a possible implementation of the function `f1` such that: [4 marks]

```
f1(a) => [1, 2, 3, 4, 100, 6, 7, 8, 9, 10]
f1(b) => [10, 9, 8, 7, 6, 100, 4, 3, 2, 1]
f1(c) => [1, 0, 1, 0, 1, 0, 2, 0, 2, 0]
f1(d) => [100, 1, 1, 1, 100]
f1(e) => [-1, 1, -2, 2, -1, 1]
```

B. Give a possible implementation of the function `f2` such that: [4 marks]

```
f2(a) => [2, 4, 100, 6, 8, 10]
f2(b) => [10, 8, 6, 100, 4, 2]
f2(c) => [0, 0, 0, 2, 0, 2, 0]
f2(d) => [100, 100]
f2(e) => [-2, 2]
```

C. Give a possible implementation of the function `f3` such that: [4 marks]

```
f3(a) => [4, 3, 2, 1]
f3(b) => [4, 3, 2, 1]
f3(c) => [2, 2, 1, 1, 1, 0, 0, 0, 0, 0]
f3(d) => [1, 1, 1]
f3(e) => [2, 1, 1, -1, -1, -2]
```

D. Give a possible implementation of the function `f4` such that: [4 marks]

```
f4(a) => [4, 3, 2, 1]
f4(b) => [4, 3, 2, 1]
f4(c) => [2, 1, 0]
f4(d) => [1]
f4(e) => [2, 1, -1, -2]
```

E. Give a possible implementation of the function `f5` such that: [4 marks]

```
f5(a) => [1, 3, 5, 7, 9]
f5(b) => [10, 8, 6, 4, 2]
f5(c) => [1, 1, 1, 2, 2]
f5(d) => [5, 1, 5]
f5(e) => [-1, -2, -1]
```

Question 3: We Love Money [18 marks]

In this question, we will explore variations of the *Count Change* problem that we discussed in Lecture 3. The code for Lecture 3 is reproduced in the Appendix for your convenient reference.

A. Write a variant of `count_change` that takes in an amount of money (in cents) and a list of coin denominations (in cents) and returns the number of ways that we can change the amount of money. For example, [5 marks]

```
count_change(100,[1,5,10,20,50]) => 343
count_change(10,[1,5]) => 3
count_change(13,[1]) => 1
count_change(10,[1,5,10]) => 4
count_change(1,[5]) => 0
count_change(20,[1,10]) => 3
count_change(5,[1,5]) => 2
```

B. Write a variant called `count_change_limited` that takes in an amount of money (in cents) and a dictionary of coin denominations (in cents) to number of coins and returns the number of ways that we can change the amount of money with the available coins. A dictionary `{1:4,5:10}` means that there are 4 1-cent coins and 10 5-cent coins available. [7 marks]
Sample execution:

```
count_change_limited(100,{1:30,5:20,10:5,20:10,50:1}) => 203
count_change_limited(10,{1:10,5:10}) => 3
count_change_limited(13,{1:100} => 1
count_change_limited(10,{}) => 0
count_change_limited(10,{1:5,5:1}) => 1
count_change_limited(10,{1:5,5:2}) => 2
count_change_limited(10,{1:10,5:3}) => 3
```

C. Ben Bitdiddle wants to improve the performance of `count_change` by memoizing the amount and number of denomination of coins that can be used. Naturally, he tried to do the same for `count_change_limited`, i.e. by memoizing the amount and number of denomination of coins that can be used. It turns out that his implementation sometimes works, but it sometimes also fails. Suggest why this might be the case. [3 marks]

D. Can memoization or dynamic programming be applied to improve the performance of `count_change_limited`? If it is possible, briefly describe how it can be done. If it is impossible, explain. [3 marks]

Question 4: Matrix Deja Vu [17 marks]

In Recitation 7, we discussed the implementation of dense and sparse matrices (see code in Appendix). In this problem, we will investigate the implementation of such matrices in OOP. We shall assume that the input used to construct the matrices is a two-level list. For example, the list `[[ 1, 2, 3], [4, 5, 6], [7, 8, 9]]` would be used to represent the input data for the following 3×3 matrix:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

`m.get_data()`, where `m` is a matrix, will return the matrix data in the same format. You can assume that the indices for our matrices start from 0 (to keep this simple).

The following is the partial implementation of a sparse matrix `SparseMatrix`.

```
class SparseMatrix:
    def __init__(self,seq):
        self.data = [len(seq),len(seq[0]),[]]
        for i in range(len(seq)):
            for j in range(len(seq[0])):
                if seq[i][j] != 0:
                    self.data[2].append([i,j,seq[i][j]])

    def rows(self):  # Returns the number of rows
        <T1>

    def cols(self):  # Returns the number of columns
        <T2>

    def get(self,x,y):
        for record in self.data[2]:
            if record[0] == x and record[1] == y:
                return record[2]
        return 0

    def set(self,x,y,val):
        for record in self.data[2]:
            if record[0] == x and record[1] == y:
                record[2]=val
                return
        self.data[2].append([x,y,val])

    def transpose(self):
        for record in self.data[2]:
           record[0],record[1]=record[1],record[0]
        self.data[0],self.data[1]=self.data[1],self.data[0]
```

```
    def get_data(self):  # Returns a list of lists representing
       <T3>              # the matrix data according to input format.
```

A. Explain how data is stored in the internal representation of `SparseMatrix`. [3 marks]

B. Complete the implementation of `SparseMatrix` by giving possible implementations for the terms `T1`, `T2`, and `T3`. [6 marks]

`T1`:
[1 marks]

`T2`:
[1 marks]

`T3`:
[4 marks]

The following is the partial implementation of a dense matrix `DenseMatrix`. Note that unlike the example discussed in Recitation, the internal representation of `DenseMatrix` uses a tuple of tuples.

```
class DenseMatrix:
    def __init__(self,seq):
        self.data = ()
        for row in seq:
            self.data += (tuple(row),)

    def rows(self):
        return len(self.data)

    def cols(self):
        return len(self.data[0])

    def get(self,x,y):
        return self.data[x][y]

    def set(self,x,y,val):  # Sets the value at (x,y) to val
       <T4>

    def transpose(self):    # transposes this matrix
       <T5>

    def get_data(self):
        return self.data
```

C. Complete the implementation of `DenseMatrix` by giving possible implementations for the terms `T4` and `T5`. [8 marks]

`T4:`
[4 marks]

T5:
[4 marks]

Question 5: Polymorphic Matrices [16 marks]

It turns out that it is too much trouble to have to decide on which implementation to use when dealing with real data, so Ben Bitdiddle decides to implement a new matrix class `Matrix`. `Matrix` can adopt either `SparseMatrix` or `DenseMatrix` (from Question 4) as the underlying matrix implementation and will switch between the two implementations transparently. The following is the partial implementation of `Matrix`:

```
class Matrix:
    def __init__(self,seq):
        def choose_sparse(seq):
            <T6>

        if choose_sparse(seq):
            self.mat = SparseMatrix(seq)
        else:
            self.mat = DenseMatrix(seq)

    def rows(self):
        return self.mat.rows()

    def cols(self):
        return self.mat.cols()

    def get(self,x,y):
        return self.mat.get(x,y)

    def set(self,x,y,val):
        self.mat.set(x,y,val)

    def transpose(self):
        self.mat.transpose()

    def get_data(self):
        return self.mat.get_data()

    def convert(self):  # swap internal representation
        <T7>
```

You should refer to Question 4 for the source codes for `SparseMatrix` and `DenseMatrix`.

A. Suppose that a tuple or list of x elements will take up x units of storage and an integer will take up 1 unit of storage. Suppose all the elements in an $m \times n$ (m rows by n columns) matrix are all non-zero and distinct integers, derive the amount of storage required for `SparseMatrix` and `DenseMatrix`. **For this question, you may assume that all integers stored are distinct and there is no memory sharing for simplicity.** [4 marks]

B. Suppose `Matrix` will choose the internal representation that minimizes total storage, complete the implementation of `Matrix` by giving a possible implementation for the term `T6`. [4 marks]

`T6:`
[4 marks]

C. `Matrix` supports an additional method called `convert` that will change the internal representation from `SparseMatrix` to `DenseMatrix`, and vice versa. For example, if the current internal representation of `Matrix` is `SparseMatrix`, calling `convert()` will cause the representation to be changed to `DenseMatrix`. Complete the implementation of `convert` by giving a possible implementation for the term `T7`. [4 marks]

`T7`:
[4 marks]

D. The current implementation of `Matrix` will decide on the internal representation when the matrix is first created. However, it would be better if the internal representation can evolve as a matrix is modified. Sketch how you would modify the current implemenation of `Matrix` so as to achieve this. You are not expected to write code to answer this question, but if you think it would be easier just to write the code instead of explaining, you are welcome to do so. [4 marks]

Question 6: 42 and the Meaning of Life [5 marks]

What are the **three** most important concepts that you learnt this past semester in CS1010S? Briefly explain the concepts and justify your answer, i.e. that they are more important than the "other" concepts.

Appendix

The following are some functions that were introduced in class:

Count Change [Lecture 3]

```
def cc(amount, kinds_of_coins):
    if amount == 0:
        return 1
    elif amount < 0 or kinds_of_coins == 0:
        return 0
    else:
        return cc(amount, kinds_of_coins-1)
             + cc(amount - first_denomination(kinds_of_coins), kinds_of_coins)

def first_denomination(kinds_of_coins):
    if kinds_of_coins == 1:
        return 1
    elif kinds_of_coins == 2:
        return 5
    elif kinds_of_coins == 3:
        return 10
    elif kinds_of_coins == 4:
        return 20
    elif kinds_of_coins == 5:
        return 50
```

Dense Matrix [Recitation 7]

```
def make_matrix(seq):
    mat = []
    for row in seq:
        mat.append(list(row))
    return mat

def rows(mat):
    return len(mat)

def cols(mat):
    return len(mat[0])

def get(mat,x,y):
    return mat[x][y]

def set(mat,x,y,val):
    mat[x][y] = val

def transpose(mat):
    transposed = []
    for i in range(len(mat[0])):
        column = []
        for j in range(len(mat)):
            column.append(mat[j][i])
        transposed.append(column)
    mat.clear()
    for row in transposed:
        mat.append(row)
    return mat

def print_matrix(mat):
    for row in mat:
        print(row)
```

Sparse Matrix [Recitation 7]

```
def make_matrix(seq):
    data = []
    for i in range(len(seq)):
        for j in range(len(seq[0])):
            if seq[i][j] != 0:
                data.append([i,j,seq[i][j]])
    return [len(seq),len(seq[0]),data]

def rows(mat):
    return mat[0]

def cols(mat):
    return mat[1]

def get(mat,x,y):
    for record in mat[2]:
        if record[0] == x and record[1] == y:
            return record[2]
    return 0

def set(mat,x,y,val):
    for record in mat[2]:
        if record[0] == x and record[1] == y:
            record[2]=val
            return
    mat[2].append([x,y,val])

def transpose(mat):
    for record in mat[2]:
       record[0],record[1]=record[1],record[0]
    mat[0],mat[1]=mat[1],mat[0]
    return mat

def print_matrix(mat):
    temp = []
    zeros = [0]*mat[1]
    for row in range(mat[0]):
        temp.append(list(zeros))
    for record in mat[2]:
        temp[record[0]][record[1]] = record[2]
    for row in temp:
        print(row)
```

— E N D   O F   P A P E R —

Scratch Paper

– H A P P Y   H O L I D A Y S ! –