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])
```

```
[[3], [2], [3]]
```

This question tests if the student understands the `is` operator and also that `list.copy()` only performs shallow copy.

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!")
```

```
Whoops
```

This question tests if the student understands the syntax for exception handling.

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)
```

```
[1, 4, 3, 2]
```

This question tests if the student understands local scope and aliasing.

D. [4 marks]

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

```
ho!  ho!  ho!  ho!  xmas!
```

This question tests if the student understands the `*args` syntax and tuple splicing.

-1 point for three `ho!`'s instead of 4.

E. [4 marks]

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

```
2
```

This question tests if the student understands the `while` loop. This question is modelled after one of the midterm questions that quite a few students got wrong.

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))
```

```
16
```

This question tests if the student understands the `*args` syntax and is able to apply nested functions.

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]
```

This question is designed to test the student's familiarity with list processing, i.e. manipulating lists either directly or using the common operators `map` and `filter`. The student first needs to figure out what needs to be done and then proceed to figure out how to do it.

```
def f1(seq):
    return list(map(lambda x:  100 if x==5 else x, seq))
```

Quite a number of students gave the following **wrong** answer:

```
def f1(seq):
    for i in seq:
       if i==5:
          i = 100
    return seq
```

Another common mistake that students make is to iterate through the index while simultaneously removing items from a list, i.e. something like:

```
    for i in range(len(seq)):
       ...
          seq.remove(seq[i])
```

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]
```

```
def f2(seq):
    return list(filter(lambda x:  x%2==0, f1(seq)))
```

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]
```

```
def f3(seq):
    temp = list(seq)
    temp.sort(reverse=True)
    return list(filter(lambda x:  x<5, temp))
```

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]
```

```
def f4(seq):
    result = []
    for item in f3(seq):
        if item not in result:
            result.append(item)
    return result
```

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]
```

```
def f5(seq):
    return seq[::2]
```

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
```

```
def count_change(amount, coins):
    if amount < 0 or len(coins) == 0:
       return 0
    elif amount == 0:
       return 1
    else:
       return count_change(amount, coins[1:])  \
            + count_change(amount - coins[0], coins)
```

The solution to this problem should not make any assumptions about the ordering of the denomination list, i.e. the solution should work even if the denomination list is not sorted.

One common mistake made is the failure to copy the list before the recursive call. Note that splicing, i.e. `coins[1:]` makes a new copy. `del coins[-1]` or `coins.remove(max_coin)` will modify the original `coin` list. It is possible to use just one list and modify it, but it has to be done very carefully.

Some students answered this question by converting it to either the original `cc` solution discussed in lecture or by implementing a DP solution. While the goal is this question was to test list copying and splicing, alternate approaches are ok.

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
```

```
def count_change_limited(amount, coins):
    if amount < 0 or len(coins) == 0:
       return 0
    elif amount == 0:
       return 1
    else:
       first_coin = tuple(coins.keys())[0]
       new_coins = dict(coins)
       new_coins[first_coin]= new_coins[first_coin] - 1
       less_coins = dict(coins)
       del less_coins[first_coin]
       if coins[first_coin] > 0:
          return count_change_limited(amount, less_coins) \
               + count_change_limited(amount - first_coin, new_coins)
       else:
          return count_change_limited(amount, less_coins)
```

The most common mistake is the failure to duplicate the `coins` dictionary correctly. The other common mistake is forgetting to check whether the number of coins has been reduced to zero.

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]

If we were to memoize only the amount and the number of denominations, it would work if the available coins was not a limited factor. If not, then we might sometimes remember the wrong value.

Credit is also given for students who can think out of the box and suggest plausible reasons why Ben's code is broken. For example, there is sharing between `count_change` and `count_change_limited` in the memoization table or maybe the base cases are wrong. Reason must be plausible.

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]

No, it is not possible to improve performance. The reason for this is that memoization and dynamic programming are about to improve performance when there are repetitions in the computation. Unfortunately, when the number of coins for each type is limited, we have to keep track of the number of coins and there are no longer any repeats in the computational state. We can apply memoization or dynamic programming (and still get the right answers), but we're not going to do any better.

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]

This part of the question and the next part (B) are designed to test that the student actually understood the material covered in Recitation 7.

The data is stored as a tuple of three items: (i) number of rows, (ii) number of columns, and (iii) matrix data. The matrix data is a list of records with the following attributes: (i) x value, (ii) y value, and (iii) data value, for the non-zero matrix entries.

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

`T1`: [1 marks]

```
return self.data[0]
```

`T2`: [1 marks]

```
return self.data[1]
```

`T3`: [4 marks]

```
temp = []
zeros = [0]*self.cols()
for column in range(self.rows()):
    temp.append(list(zeros))
for record in self.data[2]:
    temp[record[0]][record[1]] = record[2]
return temp
```

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]

```
temp = list(self.data)
temp_row = list(self.data[x])
temp_row[y] = val
temp[x] = tuple(temp_row)
self.data = tuple(temp)
```

This question is designed to test that the student understands how to modify the internal representation when it is of an immutable type.

-1 point for `return`ing the tuple instead of setting `self.data`.

```
transposed = ()
for i in range(len(self.data[0])):
    column = ()
    for j in range(len(self.data)):
        column += (self.data[j][i],)
    transposed += (column,)
self.data = transposed
```

T5:
[4 marks]

This question is designed to test that the student understands how to work with tuples.

Since the question was not well-defined, no penalty for returning a tuple instead of updating `self.data`.

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]

This question is meant to show the students another application of this idea of applying layering to hide complexity from the user. While in generic operators, we show that we can use a set of generic operators to hide different representations, we use a new class to do the same here.

Let's consider the sparse matrix. It uses length 3 tuple for row, column, and the data tuple at the highest level. The data tuple has mn elements, each of these has size one. Each a record which is a 3-element tuple which takes 6 units of memory each. Total storage = $5 + mn + 6mn = 5 + 7mn$.

Next, let's consider the dense matrix. It has a tuple of m rows. Each row is a tuple of size n, with each element pointing to one integer. Total storage = $m + m \times 2n = (2n+1)m$.

Partial credit was awarded generously for answers that are close to the right answers, i.e. $3 + 7m$ will get 1 out of 2 points for the sparse matrix and $2mn$ will get 1 out of 2 points for the dense matrix, etc.

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]

Following from the previous question, if x is number of non-zero elements, we want $5 + 7x < (2n+1)m$, so:

```
non_zeroes = 0
for row in seq:
    for element in row:
        if element !=0:
            non_zeroes += 1
rows = len(seq)
cols = len(seq[0])
return 5+7*non_zeroes < (2*cols+1)*rows
```

Basically, 2 points are given for counting zeroes and 2 points are given for checking the correct conditions. If there minor mistakes in the condition in Part(A), points are not taken off twice for the same mistake.

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]

This question tests that the student knows `isinstance`.

```
if isinstance(self.mat, DenseMatrix):
    self.mat = SparseMatrix(self.mat.get_data())
else:
    self.mat = DenseMatrix(self.mat.get_data())
```

Full credit is given if the student uses some other test to check for the type of the matrix instead of `isinstance`, but the test must be correct. Checking whether `self.mat` is a tuple is wrong. Typically, will need to check `self.mat.data`. Checking whether the `len(self.mat.data)` is equal to 3 isn't quite right since the dense matrix might have 3 columns.

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]

We can take one of two approaches. In both approaches, we need to modify the `set` method.

1. We can add a variable in `Matrix` to keep track of the number of non-zero elements. If `set` causes the condition $5+7x > (2n+1)m$ to become true, we call `convert` to swap the underlying representation.

2. We can count the number of zero elements every time after `set` is called and dynamically decide whether we want to change the underlying representation or not.

Note that `transpose` does not modify the values stored. Many students seem to suggest that it does. No points were taken off for this minor transgression.

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.

The student will be awarded points as long as he/she is coherent and doesn't say something obviously not entirely correct (which is surprisingly common because students often get stunned when they see this question and don't really know how they should deal with it). :-) One point is avoided per right concept and 2 points is awarded for coherent explanations and some valiant attempt at justifying the 3 cited concepts as most important. Actually, the "most important" isn't too critical. Just say something reasonable and not incorrect. Surprisingly some students will cite a concept and then describe the wrong thing. Obviously, some points will be taken off for "anyhow whacking."

Managing complexity is actually not exactly a concept, but the underlying theme of CS1010S, but most students will also say things like breaking down problems into smaller parts, which is fine. Concepts are things like recursion, iteration, order of growth, memoization, etc.

The goal of this question is to check that the student has actually learnt something in CS1010S (and to give away some "free" points). Students who decide to be creative and do an exposition on the principles of life instead of programming will also be awarded credit, often full credit.

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 ! –