

NATIONAL UNIVERSITY OF SINGAPORE**SCHOOL OF COMPUTING****FINAL EXAMINATION FOR****CS1010S Programming Methodology**

April 2014

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **FOUR (4)** questions and comprises **FOURTEEN (14)** printed pages. Answer **ALL** questions.
2. Read **ALL** questions before you start.
3. Write your answers in the space provided. If you need more space, write at the back of the sheet containing the question only. **DO NOT** put part of the answer to one problem on the back of the sheet containing another problem.
4. Please write **LEGIBLY!** No marks will be given for answers that cannot be deciphered.
5. This is a **CLOSED BOOK** exam. You are allowed to bring two A4 sheets of notes (written or printed on both sides).
6. Please fill in your **Matriculation Number** below.

Matriculation Number:

--	--	--	--	--	--	--	--	--	--

For Examiner's Use Only		
	Marks	Max. Marks
Question 1		30
Question 2		26
Question 3		24
Question 4		20
TOTAL:		100

Problem 1 (30 marks)

Answer each part below independently and separately. In each part, one or more Python expressions are entered into the interpreter (Python shell). The shell prompt “>>>” is omitted on each line. Determine the output printed or returned by the interpreter for the final expression entered. If the interpreter produces an error message, or enters an infinite loop, explain why.

a) [5 marks]

```
class X:
    def f(self):
        print('Good')
        return 'X'
    def g(self):
        return self.f()

class Y(X):
    def f(self):
        return 'Y'

x = X()
y = Y()
print(y.g())
```

b) [5 marks]

```
def top():
    x = [0]
    def next(*args):
        op = args[0]
        if op == 'add':
            x[0] = x[0] + args[1]
            return x[0]
    return next

a = top()
a('add', 5)
a('add', 2)
```

c) [5 marks]

```
x = [4, 3, 2, 1]
y = [1, 2, 3]

def mesh(x):
    x = ()
    y.sort(key=lambda x: -x, reverse=True)
    for i in y:
        x += (i,)
    return x

x = mesh(x)
print(x)
```

d) [5 marks]

```
def test(p, q):
    x = 'I am happy!'
    y = "I am sad!"

    try:
        if p%q == 0:
            print(x)
        else:
            print(y)

    except ZeroDivisionError:
        print("Cannot lah!")
    except TypeError:
        print("Uh oh ...")

test(10, 'Hello')
```

e) [5 marks]

```
keys = ('a', 'b', 'c', 'd')
x = {'a':1, 'b':2, 'c':3, 'd':4}
y = {'a':{'a':1}, 'b':{'b':4}, 'c':{'c':9}}

for k in keys:
    if k in y:
        print(y[k][k])
    elif k in x:
        print(x[k])
```

f) [5 marks]

```
def funny(x):
    if x > 2:
        print(x)
        return funny(x + 1)
    else:
        return funny(x - 1)

funny(3)
```

Problem 2 Sequences (26 marks)

At the Happy Magic School, Harry manages the inventory of magical items used in class for all the Year 1 students – the wands, the potions, and the spells. He has to make sure that all the magical items are in good condition, and sends them for repair or replacement when necessary.

The status of a magical item (i.e., a wand, a potion, or a spell) is represented by a tuple of two integers: (m, n) , where m is the **identity** of the magical item (e.g., wand_1, wand_2, potion_1, potion_5, spell_2, spell_5, etc.), and n is the number of remaining **charges** (i.e., how many times the wand can be “fired”, or the spell can be “cast”, etc.).

Curiously, the charges of a magical item can be a negative number: -1 – this means that the item was “overused” once, and needs to be repaired or recharged immediately. Otherwise, if the overused magical item is used again, the item will vanish forever!

Harry’s inventory lists are implemented as Python lists as follows:

```
wands = [(1, 1), (2, 2), (3, 3), (4, -1), (5, 5)]
potions = [(1, 1), (2, 1), (3, 1), (4, 1), (5, 5), (6, 6), (7, 7), (8, 8)]
spells = [(1, 5), (2, -1), (3, 5), (4, 0), (5, -1)]
```

You **DO NOT** have to worry about data abstraction for this problem, and can work on the sequences directly.

- a) Help Harry define a function `check_charges` to display the charges of the magical items, omitting the item identities, on any inventory list. The function takes in an inventory list and returns a new list showing only the charges (without the identities). An example run is shown below: [5 marks]

```
>>> check_charges(spells)
[5, -1, 5, 0, -1]
```

```
def check_charges(seq):
```

- b) Help Harry define a function `charge_5` to add 5 charges to each magical item (in any inventory list) with fewer than 2 current charges. The function takes in an inventory list and returns a new list showing only the updated charges (without the identities). An example run is shown below: [5 marks]

```
>>> charge_5(wands)
[6, 2, 3, 4, 5]
```

```
def charge_5(seq):
```

- c) Help Harry write a function `find_bad_items` that will identify all the magical items in an inventory list that need repair or recharge right away, i.e., those with charges: -1 . The function takes in an inventory list and returns a list of the bad items. An example run is shown below: [6 marks]

```
>>> find_bad_items(spells)
[(2, -1), (5, -1)]
```

```
def find_bad_items(seq):
```

- d) Impressed by your programming skills, Harry asks you to write a single function `update_inventory` that takes in an inventory list, and returns a function that in turn takes in another integer `n`, and adds `n` charges to all the items in the inventory list. Some example run outputs are shown below: [6 marks]

```
>>> ... < update wands with 4 charges > ...
[(1, 5), (2, 6), (3, 7), (4, 3), (5, 9)]
>>> ... < update spells with 4 charges > ...
[(1, 9), (2, 3), (3, 9), (4, 4), (5, 3)]
```

```
def update_inventory(seq):
```

- e) Suggest an expression for the term T1 below that would update all the wands in the original wands list by 4 charges (after the immediate repair). Your answer should lead to an updated definition of the wands inventory list as shown in the example run below: [4 marks]

```
>>> <T1>
>>> wands
[(1, 5), (2, 6), (3, 7), (4, 3), (5, 9)]
```

T1:

Problem 3 Sorting and Abstract Data Types (24 marks)

At the beginning of the summer holidays at the Happy Magic School, Harry's good friend, Ron is in-charge of collecting the magical items (i.e., the individual wands, potions, and spells) into a big magic box and storing it in the vault. Assume that the status of each magical item is again represented as a tuple of two integers (m , n) where m is the identity of the item, and n is the number of remaining charges, as defined in Problem 2.

Ron's inventory lists in a magic box are implemented in a Python dictionary as follows:

```
m_box = {'wands': [(1, 1), (2, 2), (3, 3), (4, -1), (5, 5)],
          'potions': [(1, 1), (2, 1), (3, 1), (4, 1), (5, 5), \
                      (6, 6), (7, 7), (8, 8)],
          'spells': [(1, 5), (2, -1), (3, 5), (4, 0), (5, -1)]}
```

- a) Help Ron define a function `sort_magical_items` to sort the magical items in the inventory lists (i.e., the collections of wands, potions, and spells) according to the number of charges remaining, in descending order. The function takes in a magic box and sorts the items in the box accordingly; the function does not need to RETURN anything, but the inventory lists in the magic box must be updated. You can use any sorting functions as introduced in class for this question (including the built-in Python sort functions), and can again work directly on the implementations. State any assumptions you make. An example run (with `m_box` as defined above) is shown below: [6 marks]

```
>>> sort_magical_items(m_box)
>>> m_box
{'wands': [(5, 5), (3, 3), (2, 2), (1, 1), (4, -1)],
 'spells': [(1, 5), (3, 5), (4, 0), (2, -1), (5, -1)],
 'potions': [(8, 8), (7, 7), (6, 6), (5, 5),
              (1, 1), (2, 1), (3, 1), (4, 1)]}
```

```
def sort_magical_items(box):
```


Worried that he may not be around when someone wants to check on the magic box status or repair or replace some of the magical items within, Ron has decided to introduce three abstract data types (ADT) for manipulating the magic box and its content: inventory lists and magical items. Assume that there are only three types of magical items of interest – wands, spells, and potions. A partial implementation of the three ADTs is as follows:

The magical_item ADT

```
def make_magical_item(id, charges):
    return (id, charges)
```

```
def get_identity(item):
    return item[0]
```

```
def get_charges(item):
    return item[1]
```

The inventory_list ADT

**# An inventory_list consists of the name of the magical item
collection (i.e., 'wands', 'spells', or 'potions') and a list of
magical_items (i.e., the individual wands, spells, or potions)**

```
def make_inventory_list(iname, items):
    return [iname, items]
```

```
def get_inventory_name(inv):
    return inv[0]
```

```
def get_inventory_items(inv):
    return inv[1]
```

**# All items are "collected" into an inventory_list, so the user does
not need to know how the inventory list is actually implemented**

```
def collect(items):
    return list(items)
```

The magic_box ADT

A magic_box consists of several inventory_lists

```
def make_magic_box():
    return {}
```

```
def add_inventory_list(box, inv):
    iname = get_inventory_name(inv)
    items = get_inventory_items(inv)
    <T2>
```

```
def all_inventory_lists(box):
    return list(box.items())
```


d) (**Caution:** This question is HARD!)

Help Ron redefine the function `sort_magical_items` in part a) above to sort the magical items in the inventory lists according to the number of charges remaining, in descending order.

- The new function, `sort_magical_items2` takes in a magic box and returns a NEW magic box with the new updated inventory lists.
- You can use any sorting functions (including the built-in Python sort functions) as introduced in class for this question.
- BUT your function definition can ONLY use the interface operations defined for the `magic_box`, `inventory_list`, and `magical_item` ADTs to access and modify the relevant objects and data. You may assume that the interface functions defined in parts b) and c) above are all working correctly.
- State any assumptions you make.
- **Hint:** Since you are NOT supposed to know what the underlying implementations are for the ADTs, you can try the following approach:
 1. “Transforming” the object(s) into data structures (e.g., sequences) that you can directly manipulate; and then
 2. “Transforming” the results back to the ADTs as defined.
 3. Note that the transformations can only use the interface operations of the ADTs!
- An example run (with the original `m_box` as defined earlier) is shown below: [8 marks]

```
>>> new_box = sort_magical_items2(m_box)
>>> new_box
{'potions': [(8, 8), (7, 7), (6, 6), (5, 5), (1, 1), (2, 1), (3, 1), (4, 1)],
 'wands':   [(5, 5), (3, 3), (2, 2), (1, 1), (4, -1)],
 'spells':  [(1, 5), (3, 5), (4, 0), (2, -1), (5, -1)]}
```

```
def sort_magical_items2(box):
```

Problem 4 Object-oriented Programming (20 marks)

During the summer holidays, Ernie, the smartest among the three friends at the Happy Magic School, wants to design some robot assistants that can keep her company and help her do her homework. She has provided a partial implementation of the `Robot` class and its two subclasses as follows:

```
class Robot:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name

    def set_name(self, new_name):
        self.name = new_name

class PlayRobot(Robot):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color
        self.fun_factor = 0

    def play(self, time):
        self.fun_factor += time
        print("New fun factor: " + str(self.fun_factor))

class StudyRobot(Robot):
    def __init__(self, name):
        super().__init__(name)
        self.skills = 0

    def teach(self, time):
        self.skills += time
        print("New skills level:" + str(self.skills))
```

- a) After taking a look at Ernie's code, Ron complains that he would not be able to change the name of a `PlayRobot` instance, as there are no methods to handle that in the `PlayRobot` class definition. Do you agree with Ron? **Briefly explain** why or why not? [4 marks]

- b) Ernie would like to define a new class `BuddyRobot`, that is both a `PlayRobot` and a `StudyRobot` (in that order). A `BuddyRobot` is special because it can sing! The input parameters to a `BuddyRobot` include its name, color, and a song (assumed to be a single string for now). A `BuddyRobot` has a method `sing` that prints out the lyrics (a string) of the song. Help Ernie complete the following definition for the class `BuddyRobot`. State any assumptions you make. [10 marks]

```
class BuddyRobot(PlayRobot, StudyRobot):
```

- c) Help Ernie define a new `teach` method for the `BuddyRobot` class by extending the `teach` method of `StudyRobot` with a new behavior – it will sing after teaching! State any assumptions you make. [6 marks]

```
def teach(self, time):
```