NATIONAL UNIVERSITY OF SINGAPORE

Semester 1, 2015/2016

CS1010S — PROGRAMMING METHODOLOGY

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **FIVE (5) questions** and comprises **EIGHTEEN (18) pages** including this cover page.
3. Weightage of questions is given in square brackets. The maximum attainable score is 100.
4. This is a **CLOSED** book assessment, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
5. Write all your answers in the space provided in this booklet.
6. **Please write your student number below.**

STUDENT NO: _____

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

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. [5 marks]

```
a = 7
b = 2
c = 4
def f(c):
    b = 8
    return g(c, a)
def g(b, c):
    return b * c
print(f(a))
```

B. [5 marks]

```
s = "happy"
d = {}
for i in range(10):
    d[s[i%5]] = i * 2
print(d)
```

C. [5 marks]

```
x = 2
y = 3
z = 4
def f(z):
    return lambda x: lambda y: x*z
print(f(y)(5)(7))
```

D. [5 marks]

```
def m(lst1, lst2):
    l = []
    for i in lst2:
        l = lst1.append(i)
    return l
print(m([1, 2, 3], [4, 5, 6]))
```

E. [5 marks]

```
def test(x):
    try:
        return x[0] / x[1]
    except ZeroDivisionError:
        print("Zero!")
    except IndexError:
        print("Index!")
    finally:
        return x[1] / x[0]
print(test((1, 0)))
```

F. [5 marks]

```
a = [[1, 2], [[3, 4], 5]]
b = a.copy()
a[0][1], b[1][0][0] = b[1][0], a[1][1]
print(a)
print(b)
```

Question 2: Football Mania [18 marks]

After a dismal performance in the last season, a football club is re-examining their current pool of players. They have compiled the information of each player in a Python dictionary with the following key, value pairs:

- `'Name'`: the name of the player
- `'Age'`: the age of the player in years
- `'Height'`: the height of the player in cm
- `'Weight'`: the height of the player in kg
- `'Beep'`: the beep fitness score of the player (an integer from 0 to 23)

You are tasked to use your Python skills to help the manager make his decisions.

A. [Warm up] First, the manager would like to know who is the oldest player. Implement the function `find_oldest` which takes as input a list of players and returns the name of the oldest player. [4 marks]

```
def find_oldest(players):
```

B. The manager would also like to know the average age of his current squad. Implement a function `avg_age` that takes as input a list of players and returns the average age. [4 marks]

```
def avg_age(players):
```

You realized after a while that the manger likes to know the average for various fields. If only Python has a built-in function like `sum`. But then again, you can implement it yourself!

C. Implement a higher-order function `average` that takes as input a sequence and a key, and returns the average of the given key. The key here works the same way as the key in `List.sort()`.

Example, the function in part B can be defined as:
`avg_age = lambda p: average(p, lambda x:x['Age'])` [4 marks]

```
def average(seq, key):
```

D. The manager is also interested in knowing the Body Mass Index (BMI) of the players. The BMI calculated from the height (m) and weight (kg) using this formula:

$$BMI = \frac{Weight\,(kg)}{(Height\,(m))^2}$$

Write a function `bmi` that takes as inputs the height (in cm) and weight (in kg) and returns the BMI. Then using this function, and the `average` function written in part C, implement `avg_bmi` that computes the average BMI from the input list of players. [6 marks]

```
def bmi(height, weight):



def avg_bmi(players):
```

Question 3: Ballot Boxes Deja Vu [25 marks]

The Elections Committee has reviewed the ballot boxes that you created previously and found a problem with the implementation. Because you had previously used tuples to model the boxes, they are immutable and new boxes with new votes were being created with every operation! This is certainly a security risk as votes can not be simply created.

In case you have forgotten, ballot boxes are used to collect the votes in an election. In addition, each ballot box also identifies the names of the contesting candidates.

To increase security, votes are no longer simply strings, but some objects that the Elections Committee has implemented using the following functions:

- `make_vote(string)` takes in a `string` which is the text written on the vote and returns a vote object.
- `read_vote(vote)` takes in a `vote` object and returns the string that is written on the vote.
- `is_vote(obj)` returns `True` if `obj` is a vote object and `False` is it is not.

You can assume that the Elections Committee has implemented these functions for you.

Your job is to implement a new ballot box supported by the following basic functions (other functions will be described later):

- `create_box(candidates)` takes as input a sequence of the candidates and returns an empty ballot box.
- `get_candidates(box)` takes as input a ballot box and returns a sequence of contesting candidates for the ballot box.
- `show_votes(box)` takes as input a ballot box and returns a sequence showing the names written on the votes, in no particular order.
- `cast_votes(box, vote)` takes in a ballot box and a vote and adds the vote to the box.

Sample Execution:

```
>>> box1 = create_box(('Lion', 'Mouse'))
>>> cast_vote(box1, make_vote('Lion'))
>>> cast_vote(box1, make_vote('Mouse'))

>>> box2 = create_box(('Lion', 'Mouse'))
>>> cast_vote(box2, make_vote('Cat'))
>>> cast_vote(box2, make_vote('Mouse'))

>>> get_candidates(box1)
('Lion', 'Mouse')

>>> show_votes(box1)
('Lion', 'Mouse')

>>> show_votes(box2)
```

```
('Cat', 'Mouse')
```

A. Your are to redesign the ballot boxes by incorporating Python Lists to avoid creating new boxes with every operation. But lists are mutable, and certain elements of a ballot box like the contesting candidates should not be changed.

Taking this into consideration, describe how you will represent a ballot box using lists and/or tuples and explain your choice. [3 marks]

B. Give an implementation for the functions `create_box`, `get_candidates`, `show_votes` and `cast_vote`. [6 marks]

```
def create_box(candidates):




def get_candidates(box):




def show_votes(box):




def cast_vote(box, vote):




```

C. Suppose the function `check_vote(box, vote)` takes in a ballot box and a vote, and returns `True` if the box contains the identical vote-object. The reason why the Elections Committee implemented votes as vote-objects now becomes clear in this example:

```
>>> my_vote = make_vote('Lion')
>>> cast_vote(box2, my_vote)
>>> show_votes(box2)
('Cat', 'Mouse', 'Lion')
>>> show_votes(box1)
('Lion', 'Mouse')

>>> check_vote(box1, my_vote)
False

>>> check_vote(box2, my_vote)
True
```

We see that `my_vote` is only found in `box2` and not in `box1` even though both boxes contains votes for "Lion".

Provide an implementation for the function `check_vote`. [4 marks]

D. As before, there is a need to transfer votes from one ballot box to another to consolidate them for counting. The function `transfer_votes` takes as inputs an arbitrary number of ballot boxes, and transfers all the votes from the boxes into the first box if the boxes have the same candidates. Otherwise, nothing is transfered.

Note that after transferring out the votes, the box will be empty as no two boxes should contain the same vote-object.

Sample Execution:

```
>>> box3 = create_box(('Lion', 'Bear'))
>>> cast_vote(box3, make_vote('Bear'))
>>> cast_vote(box3, make_vote('Lion'))

>>> transfer_votes(box1, box2, box2, box3)
>>> show_votes(box1)
('Cat', 'Mouse', 'Lion', 'Lion', 'Mouse')

>>> show_votes(box2)
()

>>> show_votes(box3)
('Bear', 'Lion')
```

Note that the votes are not duplicated even though we transferred twice from `box2`.

Please provide an implementation of the function `transfer_votes`. You may assume that the function will always be called with at least 2 boxes. [6 marks]

E. Now it is time to count the votes! Implement a function `count_votes` that takes as input a box, and returns a dictionary with the keys being the candidates and the values being the respective number of votes for the candidates contained in the box. An extra `'Spoilt'` key counts the number of spoilt votes in the box, that is the vote does not belong to any of the candidates. [6 marks]

Sample execution:

```
>>> show_votes(box1)
('Cat', 'Mouse', 'Lion', 'Lion', 'Mouse')

>>> count_votes(box1)
{'Mouse': 2, 'Lion': 2, 'Spoilt': 1}
```

Question 4: Despicable Me 2 [23 marks]

Your friend, Gru, is trying to model his minions as follows:

```
class Food:
    def __init__(self, name):
        self.name = name
        self.eaten = False

class Minion(Food):
    def __init__(self, name, num_eyes):
        super().__init__(name)
        self.num_eyes = num_eyes

    def get_name(self):
        return self.name

    def eat(self, food):
        if not food.eaten:
            print(self.name + " eats " + food.name)
        else:
            print(food.name + " is already eaten!")
            self.say("Matoka!")

    def say(self, words):
        print(self.name + " says '" + words + "'")
```

Gru then proceeds to test his code as follows:

```
>>> banana = Food('banana')
>>> kevin = Minion('kevin', 2)

>>> kevin.eat(banana)
Kevin eats Banana

>>> kevin.eat(banana)
Kevin eats Banana
```

"Something's wrong!", Gru exclaims. "How can the banana be eaten again?" Gru expects that instead of eating the banana, Kevin will instead say "Matoka!"

A. What is the error in Gru's code and how should he fix it? [3 marks]

Having fixed the error, Gru now finds a new error as follows:

```
>>> stuart = Minion('Stuart', 1)
>>> kevin.eat(stuart)
Kevin eats Stuart

>>> kevin.eat(stuart)
Stuart is already eaten!
Kevin says 'Matoka!'
```

"Oh no! Minions aren't supposed to eat each other!", Gru exclaims (again). "Kevin should instead say 'Sa la ka!' (which means 'how dare you') when he tries to eat something that is not Food."

B. Explain what is the problem and suggest how this can be solved. [6 marks]

Gru thinks that since minions either have only one eye or a pair of eyes, he can create two sub-classes `OneEyeMinion` and `TwoEyeMinion` which will create one-eyed and two-eyed minions respectively.

Example:

```
>>> bob = TwoEyeMinion("bob")
>>> carl = OneEyeMinion("carl")
```

C. Please help Gru implement these two sub-classes. [4 marks]

The PX-41 is a very dangerous mutator engineered in the top secret PX-Labs, located in the Arctic Circle. It mutates organisms as well as minions into purple monsters capable of great destruction!

The function `mutate` takes as input a minion and mutates it into a purple minion. Once mutated, purple minions are not capable of speaking and can only roar.

Example:

```
>>> mutate(kevin)
>>> kevin.say('Bello')
Kevin ROARS!
```

Fortunately, a PX-41 antidote exists, which reverses the effects of PX-41 and turns a purple minion back to normal. The function `cure` does this.

Example:

```
>>> cure(kevin)
>>> kevin.say('Bello')
Kevin says 'Bello'
```

D. Implement the functions `mutate(minion)` and `cure(minion)` and describe any modifications to the `Minion` class if needed.

You do not have to reproduce the entire code of the class, just snippets if you wish to. [6 marks]

Gru thinks it might be a good idea embrace object-oriented programming and define new subclasses for yellow and purple minions as such:

```
class YellowMinion(Minion):
    def __init__(self, name, num_eyes):
        super.__init__(name, num_eyes)
        self.color = 'yellow'

class PurpleMinion(Minion):
    def __init__(self, name, num_eyes):
        super.__init__(name, num_eyes)
        self.color = 'purple'
```

His idea is so we can change minions from yellow to purple and back again as follow:

```
>>> kevin = YellowMinion("Kevin", 2)
>>> kevin.say("Para Tu")
Kevin says 'Para Tu'

>>> mutate(kevin)
>>> type(kevin)
<class '__main__.PurpleMinion'>
>>> kevin.say("Para Tu")
Kevin ROARS!

>>> cure(kevin)
>>> type(kevin)
<class '__main__.YellowMinion'>
>>> kevin.say("Para Tu")
Kevin says 'Para Tu'
```

E. Do you think Gru's idea is feasible? If yes, please explain how it can be implemented. If no, please explain what is wrong with it. [4 marks]

Question 5: 42 and the Meaning of Life [4 marks]

Either: (a) explain how you think some of what you have learnt in CS1010S will be helpful for you for the rest of your life and/or studies at NUS; or (b) tell us an interesting story about your experience with CS1010S this semester. [4 marks]

— 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 ! –