NATIONAL UNIVERSITY OF SINGAPORE

Semester 2, 2016/2017

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 **TWENTY-TWO (22) 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 **ONE** double-sided A4 sheet of notes for this assessment.
5. Write all your answers in the space provided in this booklet.
6. You are allowed to write with pencils, as long as it is legible.
7. **Please write your student number below.**

STUDENT NO: ______________________

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

Question	Marks	Remark
Q1	/ 30	
Q2	/ 28	
Q3	/ 26	
Q4	/ 12	
Q5	/ 4	
Total	/ 100	

This page is intentionally left blank.

It may be used as scratch paper.

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 and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, **briefly explain why**. You may show your workings **outside the answer box** in the space beside the code. Partial marks will be awarded for workings if the final answer is wrong.

A. [5 marks]

```
s = "Lillypilly"
d = {}
for i in range(len(s)):
    d[s[i]] = i%5
print(d)
```

```
{'L': 0, 'i': 1, 'l': 3, 'y': 4,
 'p': 0}
```

This question tests the understanding that dictionaries contain unique keys, and that `'L'` is different from `'l'`.

B. [5 marks]

```
a = [1, 2]
b = [a, a]
c = a.copy()
c[0], a[1] = b[1], c[0]
print(a)
print(b)
print(c)
```

```
[1, 1]
[[1, 1], [1, 1]]
[[1, 1], 2]
```

This question tests the understanding of manipulating references of lists and how Python's tuple assignment updates the variables after the expression. Students should know this as this is commonly used to swap variables.

C. [5 marks]

```
def f(x):
    return lambda y: (x, y(x))
def g(y):
    return lambda x: x(y)
print(g(2)(f)(lambda x:x+1))
```

```
(2, 3)
```

This question tests the understanding of lambda expressions and difference between `( )` for tuples and function call.

D. [5 marks]

```
lst = [4, 3, 2, 1]
for i in lst.copy():
    lst.append(i + 1)
    lst.pop(i)
print(lst)
```

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

This question tests the understanding of copying a list to be used for iteration and the difference between `list.pop` and `list.remove`.

E. [5 marks]

```
n = 10
while n:
    if n % 5:
        n //= 2
    if n > 10:
        break
    else:
        n += 6
print(n)
```

Infinite loop. Because n will end up always being 12 at the start of the loop.

F. [5 marks]

```
def t(x, y):
    try:
        x[1] = x[0] + y
    except TypeError:
        print("This")
    except IndexError:
        print("is")
    except NameError:
        print("Sparta!")
    finally:
        return x[1]
print(t([1, "Madness"], "Blasphemy"))
```

```
'This'
'Madness'
2
```

This question tests the understanding of how exceptions are handled and that the assignment fails when there is an exception. Also test if they understand that the finally clause will always be executed.

Question 2: Bingo! [28 marks]

Bingo is a card game of chance where players match numbers on a given card. In the United States, Bingo is played using a 5x5 square gird of unique numbers on a card, like so:

3	23	39	56	64
2	20	35	50	72
14	27	41	52	70
8	16	45	53	61
7	30	36	47	71

Numbers will be drawn at random and matching numbers on the card will be marked, usually by crossing out. A player wins when a **complete row or column** of numbers have been crossed out. For simplicity, we **do not consider completing the diagonals as a win**.

Also note that though US Bingo is played on a 5x5 grid, your **implementations should work with a square bingo grid of any size**.

A. [Warm up] The function `make_card` takes as input a list of lists. Each element in the list represents a row of the bingo card as another list of numbers. `make_card` returns a data type that represents a card. A card can also be represented as a list of lists, like the dense matrix representation discussed in class.

For example, the card illustrated above is denoted as:

```
>>> card = make_card([[3,  23, 39, 56, 64],
                      [2,  20, 35, 50, 72],
                      [14, 27, 41, 52, 70],
                      [8,  16, 45, 53, 61],
                      [7,  30, 36, 47, 71]])
```

Provide an implementation of the function `make_card`. [4 marks]

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

This question is to test if the student remembers the lesson taught in recitation to deep copy the input list.

B. Whenever a number is drawn, the player will cross out the number if found on his card. The function `cross_out` takes as inputs a card and a number, and modifies the state of the card to reflect the crossing out of the number.

One possibility is to replace the matching number in the card with an `'X'` to indicate that is has been crossed out.

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

```
def cross_out(card, num):
    for row in card:
        for i in range(len(row)):
            if row[i] == num:
                row[i] = 'X'
```

C. To win the game, a player has to have one complete row or column crossed out. The functions `check_rows` and `check_cols` takes as input a card, and returns `True` if the card has at least one row, or one column respectively, crossed out. Otherwise, the functions return `False`.

Example:

```
>>> card = make_card([[4, 10], [3, 11]])  # create a 2x2 card
>>> cross_out(card, 4)  # [[X, 10], [3, 11]]
>>> check_rows(card)
False

>>> cross_out(card, 10)  # [[X, X], [3, 11]]
>>> check_rows(card)      # First row is crossed out
True

>>> check_cols(card)
False
```

```
>>> cross_out(card, 11)  # [[X, X], [3, X]]
>>> check_cols(card)     # Second column is crossed out
True
```

Provide an implementation for `check_rows` and `check_cols`. *Hint: Recall that you can multiply lists with an integer. It might be helpful to use as a comparison if your representation of a card contains a list.* [8 marks]

```
def check_rows(card):
    for row in card:
        if row == ['X'] * len(card):
            return True
    return False
```

```
def check_cols(card):
    for col in range(len(card)):
        if list(map(lambda x: x[col], card)) == ['X'] * len(card):
            return True
    return False
```

D. Your friend Chelsea thinks that it is redundant to write the `check_cols` function if you are able to transpose the card like a matrix. Her rationale is that after the transposition, the columns will becomes rows after which she can then use the `check_rows` function.

Chelsea wants to have an "in-place" transpose function, i.e., it takes constant space ($O(1)$) and does not require creating any new objects. Her idea is that since the card is always a square, she can simply swap the elements diagonally like so:

```python
def transpose(card):
    for row in range(len(card)):
        for col in range(len(card)):
            card[row][col], card[col][row] = card[col][row], card[row][col]
```

She then uses a function `print_card` that displays the card in a human-readable form to test her function:

```
>>> print_card(card)
3,  23, 39, 56, 64
2,  20, 35, 50, 72
14, 27, 41, 52, 70
8,  16, 45, 53, 61
7,  30, 36, 47, 71

>>> transpose(card)

>>> print_card(card)
3,  23, 39, 56, 64
2,  20, 35, 50, 72
14, 27, 41, 52, 70
8,  16, 45, 53, 61
7,  30, 36, 47, 71
```

Oh no! Her card did not end up getting transposed. Please explain what is wrong with Chelsea's transpose function? [4 marks]

Because her function iterates through every row and column, it swaps each pair of numbers twice, e.g. (1,2) swaps with (2,1), then (2,1) swaps again with (1,2). Thus, the result is the same as before.

A common mistake is to mention that her function has no return statement. This is not wrong because the function modifies the card and does not need to return anything.

E. Help Chelsea fix her code by providing a correct implementation of `transpose` that still fulfills her requirement of being in-place. [4 marks]

```python
def transpose(card):
    for row in range(len(card)):
        for col in range(row):
            card[row][col], card[col][row] = card[col][row], card[row][col]
```

F. Chelsea also wrote this simple function to draw the numbers for her bingo game:

```python
from random import *
def draw_number(n):
    return randint(1, n)  # Draws a number between 1 and n (inclusive)
```

However, her players complained that the numbers already drawn should not be drawn a second time. In other words, the function should not produce repeated numbers for each bingo game.

You know this can be done by creating a class which can store a list of numbers from which to draw. Each draw will only choose from the numbers in the list and then the chosen number is removed, thus ensuring that no repeated numbers will be drawn. When all possible numbers have been drawn, the draw will always return 0.

A class `Drawer` may be implemented and used like so:

```
>>> drawer = Drawer(75)  # valid numbers are from 1 to 75 inclusive.
>>> drawer.draw()
42

>>> drawer.draw()
37

>>> for i in range(73):
...     drawer.draw()
# 73 unique numbers are printed

>>> drawer.draw()
0    # all numbers have been drawn
```

Provide an implementation of the class `Drawer`.

Hint: The `random` *package contains the function* `randint(a, b)` *which returns a random integer N such that $a \leq N \leq b$. You may assume it has been imported for your use.* [4 marks]

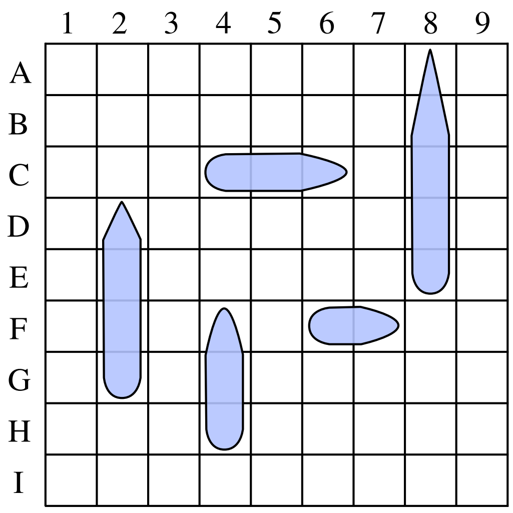
```python
from random import *

class Drawer:
    def __init__(self, n):
        self.numbers = list(range(1, n+1))

    def draw(self):
        if self.numbers:
            n = randint(0, len(self.numbers)-1)
            return self.numbers.pop(n)
        else:
            return 0
```

Question 3: Battleship [26 marks]

Battleship is another game that is played on a square grid. Players will place a fleet of ships within their grid.



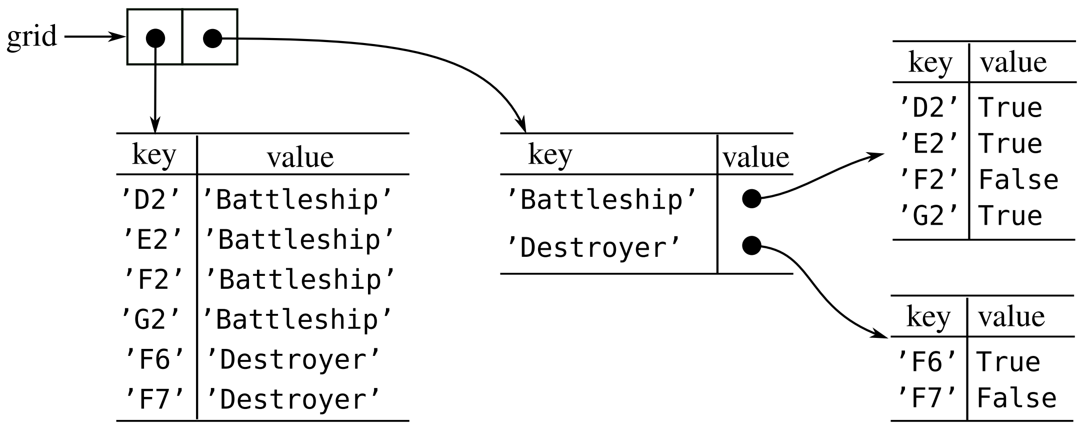
As illustrated in the above diagram, the ships have different lengths and occupies different number of points on the grid.

After a player secretly places their ships, the opponent will fire a “shot” at a point in the grid. If a ship occupies that point, that particular part of the ship is destroyed. When every occupied point of a ship is destroyed, the entire ship is sunk. Players take turns to fire shots at each other until a player’s entire fleet is sunk.

The grid of a battleship game can be represented using a **tuple** containing two **dictionaries**:

1. The keys of the first dictionary are grid points (which are strings) that are currently occupied and the value is the name (which is a string) of the occupying ship.
2. The keys of the second dictionary are the names of the ships and the respective value is a ship represented also as a dictionary. The keys of the ship dictionary are points and the values are boolean, with `False` indicating that the particular point has been destroyed and `True` otherwise.

The illustration below shows the data structure with some example data:



key	value
'D2'	'Battleship'
'E2'	'Battleship'
'F2'	'Battleship'
'G2'	'Battleship'
'F6'	'Destroyer'
'F7'	'Destroyer'

key	value
'Battleship'	●
'Destroyer'	●

key	value
'D2'	True
'E2'	True
'F2'	False
'G2'	True

key	value
'F6'	True
'F7'	False

Thus, an empty playing grid is created with the following function:

```python
def make_empty_grid():
    return ({}, {})
```

A. Rachel thinks this will not work because the grid is represented using a tuple, and since tuples are immutable, we will not be able to add points and ships to the grid. She believes we should use a list instead.

Explain why her thinking is right or wrong. [2 marks]

Rachael's thinking is wrong because though the tuple is immutable, its elements are dictionaries which are mutable. It is the references of the tuple that cannot be changed to other objects or values that makes it immutable, not the mutability of the elements.

B. Since Battleship is also played using a grid like Bingo, give one reason why do we not use the same list-of-lists representation of a Bingo board to represent the Battleship grid? [2 marks]

There are a few reasons:

- The grid is sparse, so we can save space by not storing every single square.
- There is no relation in the rows and columns in the grid, so there is no need to maintain the row and column structure like in Bingo.
- It is faster to search the ships using a dictionary than to search through a list-of-lists looking for the ships.

C. The function `add_ship` takes as inputs a grid, a name of a ship, and one or many points (which are strings), and adds the ship into the grid at the given points.

Example:

```
>>> grid = make_empty_grid()

>>> add_ship(grid, "Battleship", "D2", "E2", "F2", "G2")
>>> add_ship(grid, "Destroyer", "F6", "F7")
```

Complete the implementation of the function `add_ship`. You may assume at least one point is given and all given points are valid. [6 marks]

```
def add_ship(grid, name, *coords):  # Assume coordinates are correct
    points, ships = grid
    ships[name] = {}
    for coord in coords:
        points[coord] = name
        ships[name][coord] = True
```

D. The function `remove_ship` takes as input a grid and a ship name, and removes the ship from the grid. If the ship does not exists in the grid, `NameError` is raised with the message `"No such ship"`.

Complete the implementation for the function `remove_ship` [6 marks]

```
def remove_ship(grid, name):
    points, ships = grid
    if name not in ships:
        raise NameError("No such ship")
    for point in ships[name]:
        del points[point]
    del ships[name]
```

E. The function `is_sunk` takes as input a ship (which is a dictionary) and returns `True` if the ship is sunk, i.e., all its occupied points has been destroyed, and `False` otherwise.

Provide an implementation for the function `is_sunk`. [3 marks]

```
def is_sunk(ship):
    return not any(ship.values())
```

or using filter

```
def is_sunk(ship):
    return tuple(filter(lambda x: x, ships.values())) == ()
```

or using iteration

```
def is_sunk(ship):
    for point in ship.values():
        if point:
            return False
    return True
```

F. Provide an implementation for the function `ships_left` takes as input a grid, and returns the number of ships that are not sunk. [3 marks]

```
def ships_left(grid):
    return len(tuple(filter(lambda x: not is_sunk(x), grid[1].values())))
```

or using iteration

```
def ships_left(grid):
    count = 0
    for ship in grid[1].values():
        if not is_sunk(ship):
            count += 1
    return count
```

G. The function `attack` takes as inputs a grid and a coordinate point on the grid. If no ship is present at that point, the string `'Miss!'` is returned. Otherwise, the occupying point of the ship is destroyed.

If the ship has sunk, the string `'<name of ship> sunk'` is returned and the ship is removed from the grid. Otherwise, the grid will be updated to reflect the ship's state and the string `'Hit!'` will be returned.

Example:

```
>>> grid = make_empty_grid()
>>> add_ship(grid, "Destroyer", "F6", "F7")
>>> attack(grid, "E5")
'Miss!'

>>> attack(grid, "F6" )
'Hit!'

>>> attack(grid, "F6")  # repeated shots will produce the same effect
'Hit!'

>>> attack(grid, "F7")
'Destroyer sunk'
```

Complete the implementation for the function `attack`. [4 marks]

```
def attack(grid, coord):
    points, ships = grid
    if coord in points:
        name = points[coord]
        ships[name][coord] = False
        if is_sunk(ships[name]):
            remove_ship(grid, name)
            return name + " sunk"
        else:
            return "Hit!"
    else:
        return "Miss!"
```

Question 4: Trains II [12 marks]

Note: Please read this question carefully as it is not the same as the re-midterm question.

Consider the following implementation where we model trains using Python classes.

```python
class Car:
    def __init__(self, weight):
        self.weight = weight

    def get_weight(self):
        return self.weight

class PassengerCar(Car):
    def __init__(self, weight):
        self.passengers = 0
        super().__init__(weight)

    def get_weight(self):
        return self.passengers * 100 + super().get_weight()
```

A `Car` models a carriage that moves on tracks and it has a weight. A `PassengerCar` is a `Car` that can take on passengers, and each passenger is assumed to add 100 units to the car's weight.

Mr. Conductor wrote some code to create a passenger car:

```
>>> p1 = PassengerCar(25000)
>>> print(p1.weight)
25000

>>> p1.passengers = 100
>>> print(p1.weight)
25000
```

"Something's not right," he exclaims. "The weight should be 35,000 because I have added 100 passengers who weight 100 units each. Why does it show 25,000?"

A. Please explain how the incorrect value was given and how Mr. Conductor can get the correct weight of the railcar. [3 marks]

Mr. Conductor did not call the `get_weight()` function which will compute the weight of the railcar with the passengers. He printed the property `weight` which is the weight of the car itself. He should use `p1.get_weight()` to get the correct value.

An `Engine` is also a `Car`, but has two additional properties: power and cars. It's partial implementation is as given:

```
class Engine(Car):
    def __init__(self, power):
        self.power = power
        self.cars = []
        super().__init__(0)

    def add_car(self, car):
        self.cars.append(car)

    def get_weight(self):
        pass  # to be implemented in 4B

    def move(self):
        return self.power >= self.get_weight()
```

An `Engine` contains a sequence of cars which it is pulling. The weight of an Engine is the sum total of the weights of all the cars it is pulling. The weight of an Engine without any cars is 0.

As can be seen from the current implementation, an Engine can only move if its power is greater or equal to its weight. Thus, its method `get_weight` should return its combined weight, which is the sum total of all its cars.

Example:

```
>>> thomas = Engine(100000)
>>> thomas.get_weight()
0

>>> p1 = PassengerCar(25000)
>>> thomas.add_car(p1)
>>> thomas.get_weight()
25000

>>> p1.passengers = 100  # passengers add 100×100 = 10,000 to the weight
>>> thomas.get_weight()
35000
```

B. Provide an implementation of the `get_weight`. You may assume that an Engine will not have another Engine in its cars. [3 marks]

```python
def get_weight(self):
    return sum(map(lambda car: car.get_weight(), self.cars))

def get_weight(self):
    weight = 0
    for car in self.cars:
        weight += car.get_weight()
    return weight
```

C. A `RailCar` is a self-powered passenger car. In other words, it is both an `Engine` and a `PassengerCar`. Thus, it is able to both pull cars as well as take on passengers.

A `RailCar`'s own weight is 0 and is determined by the sum total of all its cars (like an `Engine`), as well as by the number of passengers it has (like a `PassengerCar`). It is initialized with one input: power.

One interesting issue with Railcars is that if its power is less than its weight, `RailCar.move()` raises a `SignalFault` error.

Example:

```
>>> mrt = RailCar(50000)
>>> p2 = PassengerCar(15000)
>>> p3 = PassengerCar(15000)
>>> mrt.get_weight()       # mrt has no weight on its own
0

>>> mrt.add_car(p2)
>>> mrt.add_car(p3)
>>> mrt.get_weight()
30000                      # mrt weight now includes its cars

>>> mrt.passengers = 50    # adds 5,000 weight
>>> p2.passengers  = 60   # adds 6,000 weight
>>> p3.passengers  = 70   # adds 7,000 weight

>>> mrt.get_weight()
48000                     # = 30,000 + 5,000 + 6,000 + 7,000
>>> mrt.move()
True

>>> mrt.passengers = 100  # increases weight by 5,000
>>> mrt.get_weight()
53000

>>> mrt.move()
SignalFault
```

Provide an implementation for the class `RailCar`. You may assume that the Exception class `SignalFault` is given. [6 marks]

```
class RailCar(Engine, PassengerCar):  # 1 mark
    def __init__(self, power):        # 1 mark
        super().__init__(power)       # 1 mark
        # -1 mark for superfluous statements
        # self.passengers = 0 is ok, though not needed

    def get_weight(self):
        return self.passengers * 100 + super().get_weight()  # 1 mark

    def move(self):
        if self.power < self.get_weight(): # 1 mark
            return super().move()
        else:
            raise SignalFault  # 1 mark
```

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]

The student will be awarded points as long as he/she is coherent and doesn't say something obviously wrong.

— E N D O F P A P E R —

Scratch Paper

Scratch Paper

– H A P P Y   H O L I D A Y S ! –