NATIONAL UNIVERSITY OF SINGAPORE

Semester 2, 2014/2105

Solutions for 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**.
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 matriculation number below.**

MATRICULATION 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 = '123'
b = list(a)
if b is list(a):
    print('HA!')
    if b == list(a):
        print('HE!')
else:
    print('HU!')
```

Ans: `HU!`
This question tests if student understands the blocks in `if-else` statements, and the difference between `is` and `==` comparision.

B. [5 marks]

```
a = 640
b = 4
while a//b > 0:
    for i in range(b):
        a = a // b
print(a)
```

Ans: `2`
This question tests nested `while` and `for` loops and floor division.

C. [5 marks]

```
lst = [1, 2]
tup = (3, lst)
tup[1][0] = 'z'
print(lst)
```

Ans: `['z', 2]`
This questions tests the understanding of immutable and mutable data types and aliasing.

D. [5 marks]

```
x = 10
y = 7
def foo(z):
    def bar(y):
        return y(z+1)

    if z > x:
        return z
    else:
        return bar(foo)
print(foo(y))
```

Ans: `11`
This question tests the understanding of scoping and higher order function

E.
```
def bah(f, *args):
    d = {}
    for i in args:
        d[f], f = i, i
    return d
print(bah(1, 2, 3, 4))
```

[5 marks]

Ans: `{1: 2, 2: 3, 3: 4}` #Order can be different.
This question tests understanding of `*args` syntax and Python dictionaries.

F.
```
def boo(x, y):
    try:
        valx, valy = int(x), int(y)
        print(valx / valy)
    except ZeroDivisionError:
        print('Why zero?')
    except ValueError:
        print('Wrong Value!')
boo('ten', 0)
```

[5 marks]

Ans: `Wrong Value!`
This question tests the understanding of exception handling.

Question 2: Discount Shopping [18 marks]

An online shop Ooo10 offers discount coupons which shoppers can apply to their items.

Coupons are actually implemented as Python functions that take as input an item, and return the discounted price of the item. Since items are some abstract-data type, the function `price(item)` can be used to get the price of the item.

A. **[Warm-up]** Create a coupon `half_price` that reduces the price of the item by half. For example, if `price(item)` returns 20, then `half_price(item)` returns 10. [4 marks]

```
def half_price(item):
    return price(item) / 2
```

*Note the division should not be floor division.

B. Some coupons only work under certain conditions. Create a coupon `half_price_above_20` that reduces the price of the item by half if the original price is above 20, otherwise the original price is returned. [4 marks]

```
def half_price_above_20(item):
    if price(item) > 20:
        return price(item) / 2
    else:
        return price(item)
```

*Note the division should not be floor division.

C. Your shopaholic friend Karen wants to have more discount by stacking coupons together, that is, she wants to apply more than one coupon to an item to get more discount. She thinks that if she has a `half_price` coupon and a `twenty_percent` coupon, she can use it on an item by doing `half_price(twenty_percent(item))`.

What is the error in Karen's thinking? [4 marks]

The error is that a coupon takes as input an item and returns the price. Therefore, the return value of applying a coupon is a price, which cannot be passed into another coupon.

D. Is there any way you can help Karen acheive her objectives? State the implications of any modifications you suggest, if any. **Briefly explain** your answer. [2 marks]

Not without modifications to the price function or coupon, because coupons directly returns the discounted price, which you cannot further apply any coupons.

Changing a coupon to take in the price instead of an item allows it to be stacked, but doing so might break some special condition that a coupon has on an item, e.g. coupon only works for certain brands of products.

Full marks if student answers and explains why it is not possible, or suggest appropriate modifications to the given functions along with highlighting the possible implications.

E. Warning: This question might be challenging

Karen has a list of items and a single coupon, and wants to know which is the best item to apply her coupon to to acheive the lowest total price. Recall that the actual discount the coupon gives might differ from item to item.

Write a function `best_item(coupon, items)` that takes as inputs the coupon and a list of items, and returns **the item** on which the coupon should be applied to give the best savings. [4 marks]

The simple idea is that the best item to apply the coupon is the one that gives the most difference in the price.

```
def best_item(coupon, items):
    best_price = price(items[0]) - coupon(items[0])
    best_item = items[0]
    for item in items:
        if price(item) - coupon(item) > best_price:
            best_price = price(item) - coupon(item)
            best_item = item
    return best_item
```

OR using signal-processing method

```
def best_item(coupon, items):
    # map the items into the discount
    items = list(map(lambda x: (x, price(x) - coupon(x)), items))
    items.sort(key=lambda x: x[1], reverse=True)
    return items[0][0]
```

OR a one-liner which was an actual answer by a student

```
def best_item(coupon, items):
    return max(items, key=lambda x: price(x) - coupon(x))
```

Question 3: Fly Genetics II [26 marks]

Having successfully modelled the genetics of the common fruit fly *(Drosophila melanogaster)* in the past, the Genome Institute of Singapore (GIS) has requested for your extended support.

A fly is an abstract data-type previously defined (it is not important to know the implementation details). The function `is_male(fly)` takes as input a fly and returns `True` if the fly is male, and `False` otherwise. The function `eye_colour(fly)` takes as input a fly and returns the color of the fly's eyes as a string.

A. For this question, you are to use either of the Python built-in `map` or `filter` functions in your implementation. However, partial marks may be awarded for any correct answer that does not use either the `map` or `filter` function.

Implement the function `count_males` that takes as input a list of flies and returns the number of male files in the list, and the function `count_whites` that takes as inputs a list of flies and returns the number of files whose eye-colour is `white`. [6 marks]

```
def count_males(flies):
    return len(tuple(filter(is_male, seq)))
```

```
def count_whites(flies):
    return len(tuple(filter(lambda x: eye_color(x) == "white", seq)))
```

B. Implement a function `map2(f, lst1, lst2)` that takes as inputs a function `f`, and two lists of **equal length**, and returns a list with the function `f` applied to the elements of both `lst1` and `lst2` in the same index.

In other words, `map2(f, lst1, lst2)` returns

`[f(lst1[0], lst2[0]), f(lst1[1], lst2[1]), f(lst1[2], lst2[2]), ...]`

Note: No marks will be awarded for simply writing a wrapper to the built-in `map` *function.*
[4 marks]

```
def map2(f, lst1, lst2)
    output = []
    for i in range(len(lst1)):
        output.append(fn(lst1, lst2))
    return output
```

C. Now, let us extend `map2` to support an arbitary number of lists of different lengths in the inputs, and call this new function `mapx`. The output will similarly be a list of `f` applied to the first *n* elements across all the lists, where *n* is the length of the shortest list.

Provide an implementation for `mapx`, or explain how `map2` will have to be modified to `mapx`.

Trivia: This is actually how the built-in Python map function works! Thus, no marks will be awarded for simply writing a wrapper to the built-in `map` *function.* [4 marks]

"wrapper to the built-in `map` function" means you cannot just write

```
def mapx(fn, *seq):
    return list(map(fn, *seq))
```

But you can use map in other ways like:

```
def mapx(fn, *seq):
    result = []
    for i in range(min(map(lambda x:len(x), seq))):
        items = tuple(map(lambda x:x[i], seq))
        result.append(fn(*items))
    return result
```

or a solution without using map at all:

```
def mapx(fn, *seq):
    m = len(seq[0])       # get the min length
    for s in seq:         #
        if len(s) < m:    #
            m = len(s)    #

    # now loop through the elements
    result = []
    for i in range(m):
        items = []                # assemble each 'column'
        for s in seq:             #
            items.append(s[i])    #
        result.append(fn(*items))
    return result
```

D. The scientists would like to cross-breed a group of flies with another group for their experiments. That is, a male fly in group A will breed with no more than one female fly in group B, and vice-versa. Since the number of male and female flies in each group can be random, it is possible to have flies without a mate and hence do not breed.

The function `breed(fly1, fly2)` takes as inputs two flies of different gender, and returns a new fly which is the offspring of fly1 and fly2.

Using the function `breed`, implement a function `cross_breed` that takes as inputs two lists of flies, and returns a list containing the maximum offsprings that can be bred according to the rule mentioned above.

Hint: You may make use of the function `mapx` *defined in the previous quesiton.* [6 marks]

```
def cross_breed(grpA, grpB):
    # define a function is_female
    is_female = lambda x: not is_male(x)

    # cross-breed males in A with females in B
    output = mapx(breed,
                  tuple(filter(is_male, grpA)),
                  tuple(filter(is_female, grpB)))

    # cross-breed females in A with males in B
    output.extend(mapx(breed,
                       tuple(filter(is_female, grpA)),
                       tuple(filter(is_male, grpB))))
    return output
```

E. The scientists hypothesise that the age of the flies during breeding might have some effect on the genetic composition of the offspring. Thus, during cross-breeding, they would like the oldest flies to mate together. In other words, ***the oldest male fly in group A should mate with the oldest female fly in group B***, and the second oldest pair of male and female mate with each other, and so on.

In addition, the scientist would also like to ***group offsprings with the same average age of the parents together***. For example, a fly whose parents are 5 and 10 days old will be in the same group as a fly whose parents are 6 and 9 days old.

The function `age(fly)` takes as input a fly and returns its age in days. Implement the function `cross_breed_age(males, females)` that takes as input a list of male files and a list of female flies, and breeds the flies according to the age of the files as mentioned above. ***The function will return a dictionary*** where each key is the average age of the parents, and the respective value is a list of flies with the corresponding average age of their parents.

Note: There are several steps to this function. Partial marks will be given for each correct step if the final solution is incomplete or incorrect. [6 marks]

```
def cross_breed_age(males, females):
    # sort the files in order
    males.sort(key=lambda x: age(x), reverse=True)
    females.sort(key=lambda x: age(x), reverse=True)

    # cross-breed and produce a pair (age, fly) for each couple
    offsprings = mapx(lambda x,y: ((age(x) + age(y)) / 2, breed(x,y)),
                  males,
                  females)

    # put the pairs into the dictionary
    result = {}
    for (avg, fly) in offsprings:
        if avg not in result:
            result[avg] = []
        result[avg].append(fly)

    return result
```

Alternatively, after sorting, you can use the `cross_breed` function from the previous question, with the assumption the list of offsprings is in the order of the parents.

```
    offsprings = cross_breed(males, females)
    avg_age = mapx(lambda x,y: (age(x) + age(y)) / 2, male, females)
    result = {}
    for i in range(len(offsprings)):
        if avg_age[i] not in result:
            result[avg_age[i]] = []
        result[avg_age[i]].append(offsprings[i])
    return result
```

Question 4: How to Train Your Dragon 2 [22 marks]

Your friend Hiccup of Berk, who is still trying to learn Python, shows you an implementation of creatures as follows:

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

    def get_name(self):
        return self.name

    def breathe(self):
        print(self.name + " breathes")

class FlyingCreature(Creature):
    def __init__(self, name):
        super().__init__(name)

    def fly(self):
        print(self.get_name() + " flutters")

class FireBreathingCreature(Creature):
    def __init__(self, name, charges):
        super().__init__(name)
        self.charges = charges

    def breathe(self):
        if self.charges > 0:
            print("Whoosh! Fire!")
            self.charges -= 1
        super().breathe()

    def recharge(self, amount):
        self.charges += amount
```

A. He thinks there is an error in the code as an instance of `FlyingCreature` would not be able to breathe. Do you agree with him? **Briefy explain** why or why not and provide a possible fix if needed. [4 marks]

No, Hiccup is wrong. The breathe method is inherited from the parent Creature class.

Common misconception is that the `super().__init__` method causes the inheritance. It is `class FlyingCreature(Creature)` that specifies that `Creature` is the parent class.

B. Help Hiccup define a new class `Dragon`, that is both a `FireBreathingCreature` and a `FlyingCreature`. Since there are different species of Dragons, the inputs to the class `Dragon` should include an additional string parameter `species`. In addition, Dragons should no longer "flutter" when flying. Instead, when `fly()` is called, the output should be "<name> the <species> soars!"

Example:

```
>>> toothless = Dragon("Toothless", 2, "Night Fury")
>>> toothless.breathe()
Whoosh! Fire!
Toothless breathes

>>> toothless.fly()
Toothless the Night Fury soars!
```

[6 marks]

```
class Dragon(FireBreathingCreature, FlyingCreature):
    def __init__(self, name, charges, species):
        self.species = species
        super().__init__(name, charges)

    def get_species(self):
        return species

    def fly(self):
        print(self.get_name() + " the " + self.species + " soars")
```

C. Currently, when Dragons breathe, they will go "Whoosh! Fire!" (How cool is that?) However, they only have a limited number of times they can do so.

Add a function `recharge` to the `Dragon` class that takes as input a number of charges and increases the current charges by that amount, taking care that the maximum charge cannot exceed that which was initially created.

Example:

```
>>> toothless = Dragon("Toothless", 2, "Night Fury")
>>> toothless.breathe()
Whoosh! Fire!
Toothless breathes

>>> toothless.recharge(5)
>>> toothless.breathe()
Whoosh! Fire!
Toothless breathes

>>> toothless.breathe()
Whoosh! Fire!
Toothless breathes

>>> toothless.breathe()
Toothless breathes
```

Give an implementation of the function `recharge`, and state any other modifications to the `Dragon` class that you will need. You need not reproduce the entire code for `Dragon`. [6 marks]

```
def recharge(self, amount):
        self.charges = min(self.charges + amount, self.max_charge)
```

In addition, the following line must be added to the `__init__` function:

```
    self.max_charge = charges
```

D. Since there are only a limited species of dragons, and each species has a fixed number of initial charges, it would be easier to have dragon factories of specific species to create new dragon instances.

For example, a factory can be created by:

```
>>> GronkleFactory = create_dragon_factory(5, "Gronkle")
```

After which, new instances of Gronkles can be created by:

```
>>> meatlug = GronkleFactory("Meat Lug")
>>> seaslug = GronkleFactory("Sea Slug")

>>> meatlug.fly()
Meat Lug the Gronkle soars!

>>> seaslug.fly()
Sea Slug the Gronkle soars!
```

Provide an implementation for the function `create_dragon_factory`. [6 marks]

```
def create_dragon_factory(charges, species):
    def factory(name):
        return Dragon(name, charges, species)
    return factory
```

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) write a short snippet of code that you think will convince the examiner that you have learnt Python well and hence deserve a good grade for CS1010S; or (c) 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

– H A P P Y H O L I D A Y S ! –