CS1010S — Programming Methodology
School of Computing
National University of Singapore

Re-Midterm Test Solutions

3 Oct 2018 **Time allowed:** 1 hour 30 minutes

Instructions (please read carefully):

1. This is **an open-sheet test**. You are allowed to bring one A4 sheet of notes (written or printed on both sides).
2. The QUESTION SET comprises **FOUR (4) questions** and **TEN (10) pages**, and the ANSWER SHEET comprises of **TEN (10) pages**.
3. The time allowed for solving this test is **1 hour 30 minutes**.
4. The maximum score of this test is **75 marks**. The weight of each question is given in square brackets beside the question number.
5. All questions must be answered correctly for the maximum score to be attained.
6. All questions must be answered in the space provided in the **ANSWER SHEET**; no extra sheets will be accepted as answers.
7. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
8. Use of calculators are not allowed in the test.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).

GOOD LUCK!

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: Python Expressions [25 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, **state and explain why**. You may show your workings **outside the answer box**. Partial marks may be awarded for workings even if the final answer is wrong.

The code is replicated on the answer booklet. 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.

```python
x = 7
y = 2
def f(y):
    return y + x
def g(x, y):
    return f(y - x)
print(g(y, f(x)))
```

[5 marks]

B.

```python
a = ("o",)
b = ("b",) + (a, a)
c = b + (b + b)
print(c)
```

[5 marks]

C.

```python
s = "halloween"
while s[0] != s[-2]:
    half = len(s)//2
    s = s[half:-1] + s[1:half]
print(s)
```

[5 marks]

D.

```python
def tick(x, y):
    return lambda boo: boo(y, "tock")
def tock(y, x):
    return lambda boo: boo(tick, y)
print(tock("tick", "boom")(tick)(lambda bo,om: bo+om+"!"))
```

[5 marks]

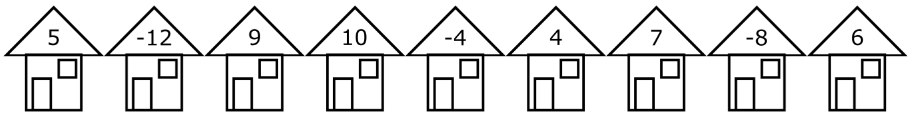
E.

```python
date = 0
if date:
    print("today")
else:
    print("happy")
if date == 31//8//2018:
    print("halloween")
elif date+1 == 1//9//2018:
    print("october")
else:
    print(final-exam-approaching)
```

[5 marks]

Question 2: Halloween [18 marks]

Halloween is time for trick-or-treat where kids go from one house to another asking for candies. Typically, you just go to consecutive houses to collect your candies but you realize that you do not like all the candies. In fact, you hate some of them. Since the house in your neighborhood always give the same candy each year, you assign a score to the house. The score for each house can be seen on the roof.



Since you can only visit consecutive houses, the following visits are valid: `(-12, 9, 10, -4)`, `(10, -4, 4)`, `(10)`, and `()`. On the other hand, the following visits are invalid: `(5, 9)`, and `(9, 10, -8)`. We let first house on the left be house #0 and let the block of houses be represented as a tuple of `int`.

The function `house_visit` takes as input a tuple of `int` called `block` (i.e., block of houses), a starting house number `num`, and the number of houses to visit `h`. The function returns the total sum of the scores of the houses you visited. If you reached the end of the block of houses and still have houses to visit, you just ignore them and go home with your candies instead. For instance, `house_visit((5, -12, 9, 10, -4, 4, 7, -8, 6), 1, 4)` returns `3`.

A. **[Warm up]** Provide an implementation of `house_visit(block, house_num, h)`. Note that you are not allowed to use the built-in function `sum`. [2 marks]

In the subsequent questions, you are allowed to call the function defined in *Part (A)*. However, do note that the order of growth in time and space depends on your actual implementation of function in *Part (A)*.

Now that you can visit the houses, you are interested in finding the best consecutive houses to visit. As you are planning an event after the trick-or-treat, you can only visit a limited number of houses. The function `good_visit` takes as input a tuple of `int` called `block` (i.e., block of houses), and the number of houses to visit `h`. The function returns the largest score when visiting exactly `h` consecutive houses. For instance, using the block of houses above, the largest score when visiting 3 consecutive houses can be obtained by visiting the 3rd to 5th houses. This will yield a score of `9 + 10 - 4 = 15`.

B. Write an iterative function `good_visit(block, h)`. [4 marks]

C. State the order of growth in terms of time and space for the function you wrote in *Part (B)*. Briefly explain your answer. [2 marks]

D. Write an recursive function `good_visit(block, h)`. [4 marks]

E. State the order of growth in terms of time and space for the function you wrote in Part (D). Briefly explain your answer. [2 marks]

F. The function `best_visit(block)` takes as input a tuple of `int`, and returns the best possible score from consecutive houses. Note that you may possibly visit all of the houses or none of the houses.

Provide an implementation for the function `best_visit`. You may use any functions you have defined previously in this question. [4 marks]

Question 3: Higher-Order Halloween [10 marks]

A. Consider the higher-order function `fold` which was taught in class.

```
def fold(op, f, n):
    if n == 0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))
```

The function `good_visit` in Question 2 can be defined in terms of `fold` as follows:

```
def good_visit(block, h):
    <PRE>
    return fold(<T1>, <T2>, <T3>)
```

Please provide possible implementations for the terms `T1`, `T2`, and `T3`. You may optionally define other functions in `PRE` if needed. Note you are to use ONLY the given higher-order function and not solve it recursively or iteratively. [4 marks]

B. **[Warning: HARD]** Your friend Jack O' Lantern claims that with clever tricks, the functions `good_visit(block, h)` and `best_visit(block)` can be solved using ONLY `map`, `max`, and `enumerate_interval` without any other iteration or recursion except those provided by the higher-order function.

```
def map(fn, seq):
    if seq == ():
        return ()
    else:
        return (fn(seq[0]),) + map(fn, seq[1:])

def enumerate_interval(low, high):
    return tuple(range(low, high+1))
```

Is your friend correct? For each function, if you say he is incorrect, explain why it is impossible. Otherwise, if you say he is correct, provide an implementation. [6 marks]

Question 4: Trick-or-Treat [22 marks]

INSTRUCTIONS: Please read the entire question clearly before you attempt this problem!! You are also not to use any Python data types which have not yet been taught in class.

Now that your planning is completed, your friend Jack O' Lantern is going with you for Trick-or-Treating. A candy has three properties: the name (`str`), the flavour (`str`), and the weight (`str`). It is supported by the following functions:

- `make_candy(name, flavour, weight)` takes the candy name, the candy flavour, and the weight as inputs, and returns a new candy.
- `name(candy)` takes a candy as input, returns its name.
- `weight(candy)` takes a candy as input, returns its weight.

Since candies are inside the wrapper, the flavour of the candy is unknown until it is opened and eaten. The implementation should be done in such a way that when a candy is printed using the function `print(candy)`, its flavour remains unknown.

A. Provide an implementation for the *candy* functions `make_candy`, `name` and `weight`.
[4 marks]

We wish to add the following functions for candy:

- `flavour(candy)` takes a candy as input, if it is not opened yet, return the string `'Unknown'` but if it is opened, returns the flavour of the candy.
- `taste(candy)` takes a candy as input, returns an opened and tasted candy.

Consider the following sample run:

```
>>> kitkat = make_candy("Kit Kat", "Chocolate", 10)
# make a 10g Kit Kat chocolate bar

>>> flavour(kitkat)         # unopened
Unknown

>>> kitkat = taste(kitkat)  # open and taste
>>> flavour(kitkat)         # opened and tasted
Chocolate

>>> kitkat = taste(kitkat)  # already opened
>>> flavour(kitkat)         # no change
Chocolate
```

B. Provide an implementation for the *candy* functions `flavour` and `taste`. Note that if your implementation does not hide the flavour when printed using `print`, the maximum mark obtained is 2.
[4 marks]

[Important!] For the remaining parts of this question, **you should not break the abstraction of *candy* in your code.**

Your treat bag is marked with your name (`str`) and can only hold a certain amount of weight (`int`). Candies can be added into the bag during trick-or-treating. The treat bag is supported by the following functions:

- `make_bag(name, max_weight)` takes your name and the maximum weight the treat bag can hold as inputs, and returns a new treat bag.
- `add_candy(candy, bag)` takes a candy and a bag as inputs, and returns a treat bag with the candy added to it *only if* the bag can still carry the candy, otherwise return the original bag. Note that it is possible to add the same candy multiple times.

Consider the following sample run:

```
>>> jackbag = make_bag("Jack O' Lantern", 21) # Jack's treat bag

>>> kitkat = make_candy("Kit Kat", "Chocolate", 10) # Kit Kat
>>> reeses = make_candy("Reese's", "Pumpkin", 5)    # Reese's

>>> jackbag = add_candy(kitkat, jackbag)
>>> jackbag = add_candy(reeses, jackbag)
>>> jackbag = add_candy(kitkat, jackbag)
>>> jackbag = add_candy(reeses, jackbag)
```

C. Draw the **box-pointer diagram** of `jackbag`, `kitkat` and `reeses` at the end of the sample run above. Your diagram should be consistent with your implementations of *candy* and *treat bag* functions in Part (B), Part (C), and Part (D). [2 marks]

D. Provide an implementation for the *treat bag* functions `make_bag` and `add_candy`. [4 marks]

At the end of trick-or-treating, You and Jack wants to know the contents of the treat bag.

E. Implement the function `get_names` which takes a treat bag as input and returns a `tuple` of all candy names. Note that if there are duplicate names, the order does not matter.

[4 marks]

F. Implement the function `get_flavours` which takes a treat bag as input and returns a `tuple` of all candy flavours, including the flavours of unopened candies. Note that if there are duplicate flavours, the order does not matter. [4 marks]

Appendix

The following are some functions that were introduced in class. For your reference, they are reproduced here.

```
def sum(term, a, next, b):
  if a > b:
    return 0
  else:
    return term(a) + sum(term, next(a), next, b)

def product(term, a, next, b):
  if a > b:
    return 1
  else:
    return term(a) * product(term, next(a), next, b)

def fold(op, f, n):
  if n == 0:
    return f(0)
  else:
    return op(f(n), fold(op, f, n-1))

def enumerate_interval(low, high):
    return tuple(range(low,high+1))

def map(fn, seq):
    if seq == ():
        return ()
    else:
        return (fn(seq[0]),) + map(fn, seq[1:])

def filter(pred, seq):
    if seq == ():
        return ()
    elif pred(seq[0]):
        return (seq[0],) + filter(pred, seq[1:])
    else:
        return filter(pred, seq[1:])

def accumulate(fn, initial, seq):
    if seq == ():
        return initial
    else:
        return fn(seq[0], accumulate(fn, initial, seq[1:]))
```

Scratch Paper

Scratch Paper

— E N D O F P A P E R —

CS1010S — Programming Methodology
School of Computing
National University of Singapore

Re-Midterm Test Solutions — Answer Sheet

3 Oct 2018 **Time allowed:** 1 hour 30 minutes

Student No:

S	O	L	U	T	I	O	N	S

Instructions (please read carefully):

1. Write down your **student number** on this answer sheet. DO NOT WRITE YOUR NAME!
2. This answer sheet comprises **TEN (10) pages**.
3. All questions must be answered in the space provided; no extra sheets will be accepted as answers. You may use the extra page at the back if you need more space for your answers.
4. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
5. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).

GOOD LUCK!

For Examiner's Use Only

Question	Marks	Remarks
Q1	/ 25	
Q2	/ 18	
Q3	/ 10	
Q4	/ 22	
Total	/ 75	

This page is intentionally left blank.
Use it ONLY if you need extra space for your answers, in which case indicate the **question number clearly**. **DO NOT** use it for your rough work.

Question 1A

[5 marks]

```python
x = 7
y = 2
def f(y):
    return y + x
def g(x, y):
    return f(y - x)
print(g(y, f(x)))
```

`19`
Tests the understanding of scoping.

Question 1B

[5 marks]

```python
a = ("o",)
b = ("b",) + (a, a)
c = b + (b + b)
print(c)
```

```
('b', ('o',), ('o',), 'b', ('o',),
 ('o',), 'b', ('o',), ('o',))
```
Test `tuple` addition

Question 1C

[5 marks]

```python
s = "halloween"
while s[0] != s[-2]:
    half = len(s)//2
    s = s[half:-1] + s[1:half]
print(s)
```

`IndexError` Test loop with index out of range

Question 1D [5 marks]

```
def tick(x, y):
    return lambda boo: boo(y, "tock")
def tock(y, x):
    return lambda boo: boo(tick, y)
print(tock("tick", "boom")(tick)(lambda bo,om: bo+om+"!"))
```

`"ticktock!"`
Test knowledge of evaluation of lambda expressions

Question 1E [5 marks]

```
date = 0
if date:
    print("today")
else:
    print("happy")
if date == 31//8//2018:
    print("halloween")
elif date+1 == 1//9//2018:
    print("october")
else:
    print(final-exam-approaching)
```

`'happy'` Test `0` is `False`.
`'halloween'` Test independent `if`-`else`.

Question 2A [2 marks]

```
def house_visit(block, num, h): # Iterative -> Time: O(n), Space: O(1)
    score = 0
    for i in range(num, num + h):
        if i < len(block):
            score = score + block[i]
    return score
def house_visit(block, num, h): # Recursive -> Time: O(n), Space: O(n)
    if h == 0 or num >= len(block):
        return 0
    else:
        return block[num] + house_visit(block, num + 1, h - 1)
```

Question 2B [4 marks]

```
def good_visit(block, h):
    max_score = 0
    for num in range(0, len(block)-h+1):
        score = house_visit(block, num, h)
        if score > max_score:
            max_score = score
    return max_score
# Alternative
def good_visit(block, h):
    score, max_score = 0, 0
    for num in range(0, h):
        score = score + block[num]
    for num in range(h, len(block)-h+1):          # basically use queue
        score = score + block[num] - block[num-h]# add back, remove front
        if score > max_score:
        max_score = score
    return max_score
```

Question 2C [2 marks]

Time: $O(n^2)$, where n = `len(block)`. `house_visit` takes $O(n)$ time (*assume implementation in solution*), and the loop iterates n times.
(Alternative) $O(n)$, where n = `len(block)`. The first loop iterates h times and the second loop also $n - h$ times for a total of n iteration.

Space:$O(1)$ if using iterative `house_visit` or *alternative* solution. No extra memory is needed as all the variables are overwritten with the new values.
$O(n)$ if using recursive `house_visit`. Because `house_visit` requires $O(n)$ space.

Question 2D [4 marks]

```python
def good_visit(block, h):
    if len(block) < h:
        return 0
    else:
        res1 = house_visit(block, 0, h)
        res2 = good_visit(block[1:], h)
        if res1 > res2:
            return res1
        else:
            return res2
```

Question 2E [2 marks]

Time:$O(n^2)$, where n = `len(block)`. `house_visit` (*or slicing*) takes $O(n)$ time, and there are n recursions.

Space:$O(n^2)$, where n = `len(block)`. There are n recursions and each recursion storing the sliced tuple of length at most n.
More specifically: $O(n)$ (*stack*) and $O(n^2)$ (*heap*).

Question 2F [4 marks]

```python
def best_visit(block):
    max_score = 0
    for h in range(1, len(block)):
        score = good_visit(block, h)
        if score > max_score:
            max_score = score
    return max_score
```

Question 3A

[4 marks]

*optional <PRE>:

<T1>:

```
lambda a, b: a if a > b else b #alternative max(a,b)
```

<T2>:

```
lambda n: house_visit(block, n, h)
```

<T3>:

```
len(s)-1
```

Question 3B

[6 marks]

```
# good_visit
def good_visit(block, h):
    return max(map(lambda num: house_visit(block, num, h),
                   enumerate_interval(0, len(block))))
```

```
# best_visit
def best_visit(block):
    return max(map(lambda h: good_visit(block, h),
                   enumerate_interval(1, len(block))))
```

Question 4A

[4 marks]

```
def make_candy(name, flavour, weight):
    return (name, weight, lambda : flavour, False)



def name(candy):
    return candy[0]



def weight(candy):
    return candy[1]
```

Question 4B

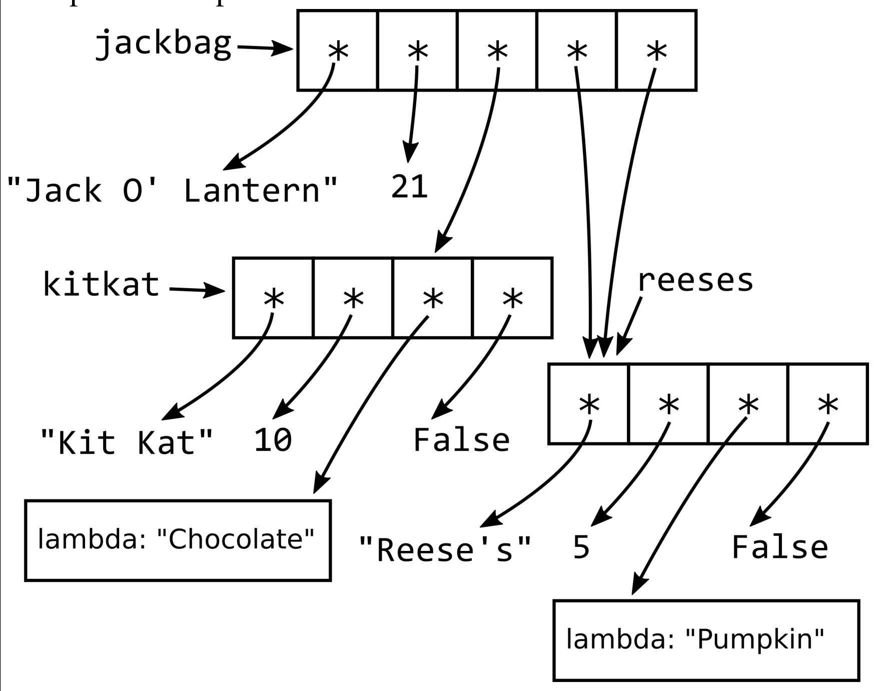
[4 marks]

```
def flavour(candy):
    if candy[3]:
        return candy[2]
    else:
        return "Unknown"



def taste(candy):
    if candy[3]:
        return candy
    else:
        return (candy[0], candy[1], candy[2](), True)
```

Question 4C

[2 marks]

One possible implementation is this:



Question 4D

[4 marks]

```python
def make_bag(name, max_weight):
    return (name, max_weight)



def add_candy(candy, bag):
    weight = sum(map(lambda c: weight(c), bag[2:])
    if weight > max_weight:
        return bag
    else:
        return (name, max_weight - weight) + bag[2:] + (candy,)
```

Question 4E

[4 marks]

```
def get_names(bag):
all_names = ()
for candy in bag[2:]:
    all_names = all_names + (name(candy),)
return all_flavours
# Using map
def get_names(bag):
    return map(name, bag[2:])
```

Question 4F

[4 marks]

```
def get_flavours(bag):
    all_flavours = ()
    for candy in bag[2:]:
        all_flavours = all_flavours + (flavour(taste(candy)),)
    return all_flavours
def get_flavours(bag):
    return map(lambda candy: flavour(taste(candy)), bag[2:])
```

— END OF ANSWER SHEET —