CS1010S — Programming Methodology
School of Computing
National University of Singapore

# Re-Midterm Test Solutions

23 March 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**. Your marks will be **capped at 38** if you are taking the test for the second time. 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.**

```
x = 7
y = 11
def f(x, y):
    return x - y
print(f(y + f(y, x), x + f(x, y)))
```

[5 marks]

**B.**

```
a = ()
b = (1)
c = (2, 3)
d = (a, b, c) + (a)
print(d)
```

[5 marks]

**C.**

```
m = '0'
n = '2'
if m:
    m *= 2
if m == n:
    m += 3
elif n:
    n = m + n
    print(n)
m *= n
print(m)
```

[5 marks]

**D.**

```
i = 8
for i in range(10):
    i += 4
    if i % 2 == 0:
        i -= 5
        continue
    if i % 3 == 0:
        break
    print(i)
```

[5 marks]

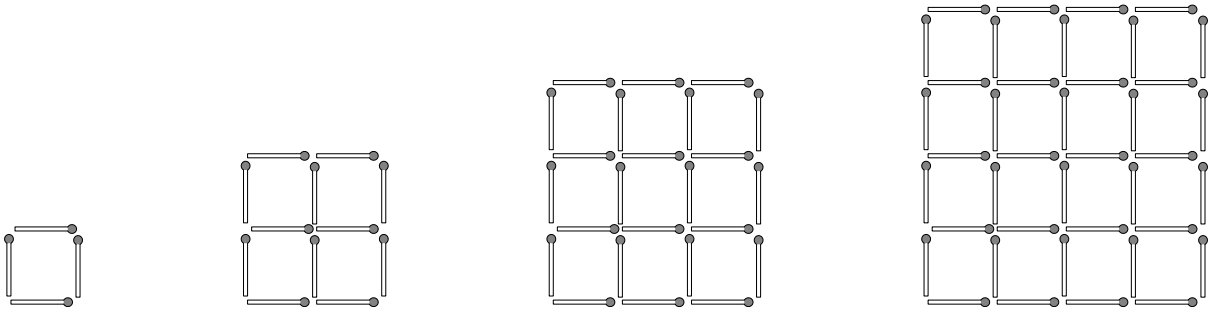
**E.**

```
def foo(x):
    return lambda y: y(y(x))
def bar(x):
    return lambda y: x(x(y))
print(foo(lambda x: x+x)(bar)(1))
```

[5 marks]

# Question 2: Matchsticks [18 marks]

Matchsticks can be used to make square grids of different sizes. The figures below show grids of size 1, 2, 3 and 4 made of 4, 12, 24 and 40 matchsticks, respectively:



The function `num_sticks` takes as input the grid size, and returns the number of matchsticks required to form the grid.

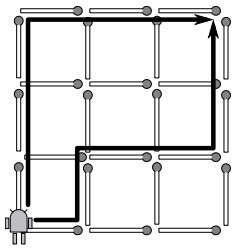
**A.** Provide a recursive implementation of the function `num_sticks`. [4 marks]

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

**C.** Provide a iterative implementation of the function `num_sticks`. [4 marks]

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

**E.** Suppose the grid represents a system of roads, and a robot wishes to travel from one corner to the opposite diagonal corner. It will always take the shortest route, i.e., only moving towards the destination. The figure below shows two such paths on a size 3 grid:



The function `num_paths` takes as input the grid size, and returns the number of unique shortest paths that the robot can take to travel from one corner of the grid to the opposite diagonal corner.

Provide an implementation for `num_paths`. [4 marks]

**F.** Is your implementation in Part (E) recursive or iterative? State the order of growth in terms of time and space of your implementation and briefly explain your answer. [2 marks]

# Question 3: Higher-Order Matchsticks [8 marks]

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

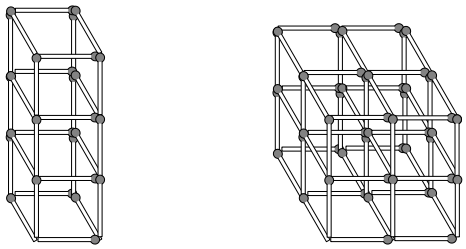
```
def fold(op, f, n):
    if n == 0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))
```

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

```
def num_sticks(n):
    <PRE>
    return fold(<S1>, <S2>, <S3>)
```

Please provide possible implementations for the terms `S1`, `S2` and `S3`. You may optionally define other **functions** in `PRE` to be used as your terms if needed. [4 marks]

**B.** We can also build a 3D tower of matchsticks by stacking up grids. The figures show a tower made from a size 1 grid of height 3 and a size 2 grid of height 2.



The higher-order function `make_stick_counter` takes as input the size of a grid, and returns a function that calculates the number of matchsticks needed to build a tower of the given size of height $h > 0$.

Example:

```
>>> num_sticks_1x1_tower = make_stick_counter(1)
>>> num_sticks_1x1_tower(3)
28

>>> num_sticks_2x2_tower = make_stick_counter(2)
>>> num_sticks_2x2_tower(2)
54
```

Provide an implementation for the function `make_stick_counter`. You do not need to solve this question recursively or iteratively and may wish to reuse the function `num_sticks` defined earlier. [4 marks]

# Question 4: Projectors [24 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.**

*A projector or image projector is an optical device that projects an image (or moving images) onto a surface, commonly a projection screen. Most projectors create an image by shining a light through a small transparent lens, but some newer types of projectors can project the image directly, by using lasers.* *Source: Wikipedia.*

For this question, you will be modelling projector and the lens. First, you will need to know about the model of an image. An image can be modelled by its height (`int`) and width (`int`). You may assume that the following code is already given for image.

```
def make_image(height, width):
  return (height, width)

def height(img):
  return img[0]

def width(img):
  return img[1]
```

**[Important!]** For all parts of this question, **you should not break the abstraction of *image* in your code.** The implementation above is given only as reference.

A lens consist of a name (`str`), and four refraction factors (`int`), $a$, $b$, $c$, and $d$. When an image of height $h$ and width $w$ is beamed through the lens, a new image of height $(h \times a) + (w \times c)$ and width $(h \times b) + (w \times d)$ will be created. In other words, the image will be distorted according to the refraction factors.

A lens is supported by the following functions:

- `make_lens(name, a, b, c, d)` takes the lens name and its refraction factors as inputs, and returns a new lens.
- `name(lens)` takes a lens as input, returns its name.
- `beam(lens, img)` takes a lens and an image as inputs, returns an image that has been distorted by the lens using the above equation.

A projector is first created with a name (`str`). Various lenses can be placed into the projector, with each new lens placed farthest away from the image source.

A projector is supported by the following functions:

- `make_projector(name)` takes the projector name as input, and returns a new projector.
- `place_lens(proj, lens)` takes a projector and a lens as inputs, and returns a projector with the lens added to it. Note that it is possible to place the same lens multiple times and that the order of placement of lenses will determine the final image that is projected.

Consider the following sample run:

```
>>> cinema = make_projector("Cinema") # make a cinematic projector

>>> upside_down = make_lens("UpsideDown", -1, 0, 0, 1)
>>> enlarge_double = make_lens("Enlarge2X", 2, 0, 0, 2)
>>> widen = make_lens("Widen", 1, 0, 0, 4)

>>> cinema = place_lens(cinema, enlarge_double)
>>> cinema = place_lens(cinema, upside_down)
>>> cinema = place_lens(cinema, enlarge_double)
>>> cinema = place_lens(cinema, widen) # made for widescreen
```

**A.** Draw the **box-pointer diagram** of `cinema`, `upside_down` and `enlarge_double` at the end of the sample run above. Your diagram should be consistent with your implementations of *lens* and *projector* functions in Part (B) and Part(C). [2 marks]

**B.** Provide an implementation for the *lens* functions `make_lens`, `name` and `beam`. [4 marks]

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

**C.** Provide an implementation for the *projector* functions `make_projector` and `place_lens`. [4 marks]

**D.** Stan Lee Kubrick loves to go to the cinema. He is also a collector of weird lenses. One of his hobbies is to watch movies on a big screen. To project a movie, an image starts from the source and is beamed through each *lens* in succession until it displays on a screen. The distortion of the final image displayed on the screen depends on the refraction factors of all the lenses in the projector.

Implement the function `get_lenses` which takes a *projector* as input, and returns a tuple of the names of each *lens* in the projector starting from the lenses closest to the image source. [4 marks]

**E.** Implement the function `projection`, which takes a *projector* and an *image* as inputs, and beams the image through the lenses placed in the projector in succession, starting with the lens closest to the image source. It returns the final *image* that will be projected on the screen. [4 marks]

**F.** Stan has another *projector* `vintage_proj` with various lenses for watching vintage films. He wants to use the lenses from `vintage_proj` together with the first projector and runs the following code:

```
>>> cinema = place_lens(cinema, vintage_proj)
```

Describe what happens when he tries to call the function `projection` on `cinema`? [6 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

23 March 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 | / 8 | |
| Q4 | / 24 | |
| **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]

```
x = 7
y = 11
def f(x, y):
    return x - y
print(f(y + f(y, x), x + f(x, y)))
```

`12` Tests the understanding of scoping and evaluation of expressions.
If answer is incorrect and no working is shown, 0 marks will be awarded.
+2 each: evaluation of `y + f(y, x)` and `x + f(x, y)`
+1: final evaluation of `f`

## Question 1B

[5 marks]

```
a = ()
b = (1)
c = (2, 3)
d = (a, b, c) + (a)
print(d)
```

`((), 1, (2, 3))` Tests understanding of tuple addition and expansion.
+1 each: `a` is an empty tuple, and `b` is an integer
+2: correct expansion of `(a, b, c)`
+1: correct addition of tuples

## Question 1C

[5 marks]

```
m = '0'
n = '2'
if m:
    m *= 2
if m == n:
    m += 3
elif n:
    n = m + n
    print(n)
m *= n
print(m)
```

`'002'` Tests implicit conversion of string to boolean and string multiplication.
`TypeError` Cannot multiply str with str. Tests understanding of independence of multiple if-else statements and runtime errors.
+1: evaluate `m` as `True`, AND evaluate `n` as `True` (if applicable)
+1: evaluate `m *= 2`, AND evaluate `n = m + n` (if applicable)
+1 each: identify independent if-statement, print `n` correctly, identify/state error `m *= n`
-4: fail to execute code due to error

## Question 1D

[5 marks]

```
i = 8
for i in range(10):
    i += 4
    if i % 2 == 0:
        i -= 5
        continue
    if i % 3 == 0:
        break
    print(i)
```

`5`
`7` Tests the understanding of scope within `for` loops, `break` and `continue`.
+1 each: correct application of continue and break
+3: correct scope and updating of `i`

## Question 1E

[5 marks]

```
def foo(x):
    return lambda y: y(y(x))
def bar(x):
    return lambda y: x(x(y))
print(foo(lambda x: x+x)(bar)(1))
```

`16` Tests knowledge of evaluation of lambda expressions.
If answer is incorrect and no working is shown, 0 marks will be awarded.
-1: careless mistake
-3: expression is partially but correctly evaluated and explained

## Question 2A

[4 marks]

```
def num_sticks(n):
    if n == 0:
        return 0
    else:
        return num_sticks(n-1) + 4*n
```

## Question 2B

[2 marks]

Time: $O(n)$, there is a total of $n$ recursive calls.

Space:$O(n)$, there is a total of $n$ recursive calls, and each call will take up space on the stack.

## Question 2C

[4 marks]

```
def num_sticks(n):
    num = 0
    for i in range(n+1):
        num += 4*i
    return num
```

## Question 2D

[2 marks]

Time: $O(n)$, the loop will iterate $n$ times.

Space:$O(1)$, no extra memory is needed because the variables are overwritten with the new values.

## Question 2E

[4 marks]

```
def num_paths(n): # recursive
    def helper(x, y):
        if x < 0 or y < 0:
            return 0
        if x == 0 and y == 0:
            return 1
        else:
            return helper(x-1, y) + helper(x, y-1)
    return helper(n, n)
```

## Question 2F

[2 marks]

Time: $O(2^n)$ because there are two recursive calls in each computation step, which results in a tree recursion. Thus, the total number of computation steps is the number of leaves in the tree which is $2^n$.

Space: $O(n)$ because each recursion call will eventually end at the destination with a path length of $2m$. So at most $O(n)$ of calls are used in memory to store the recursive calls.

## Question 3A

[4 marks]

*optional <PRE>:

This question is marked as a whole - simply filling common values in the terms will not give you marks. There has to be some understanding shown in your answer.
1 mark: Demonstrate knowledge of `fold` arguments and `S3` is correct.
2 marks: Show some understanding of `num_sticks`'s accumulative pattern.
3 marks: Careless mistake

<S1>:

```
lambda x, y: x + y
```

<S2>:

```
lambda x: 4*x
```

<S3>:

```
n
```

## Question 3B

[4 marks]

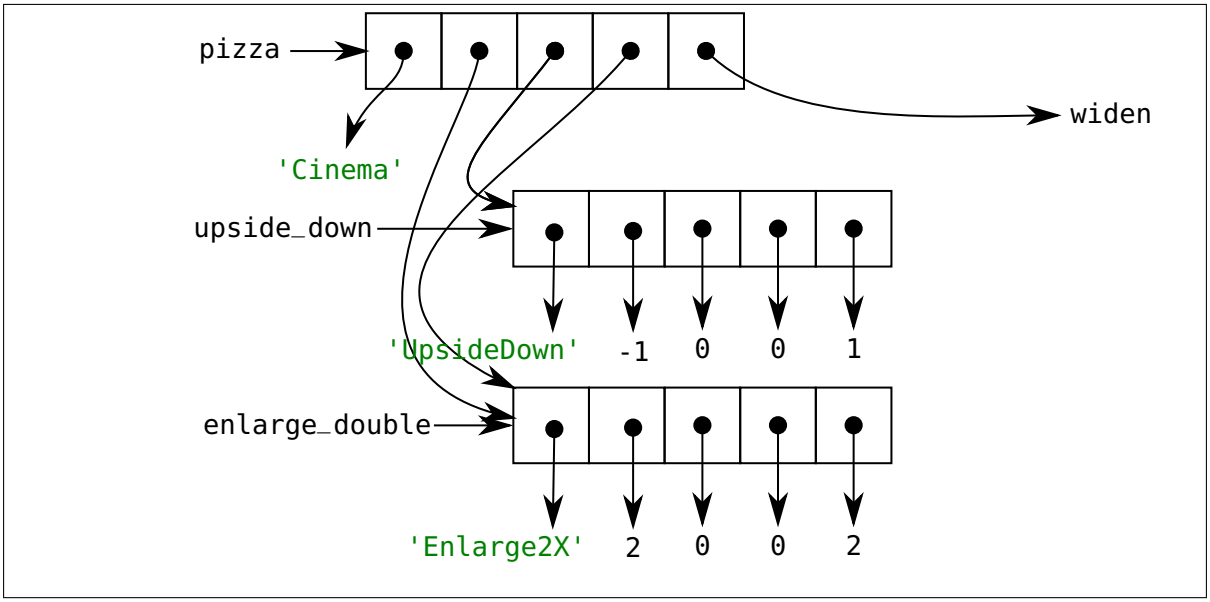
```
def make_stick_counter(n):
    return lambda h: (h+1)*num_sticks(n) + h*(n+1)**2

# recursive solution
def make_stick_counter(n):
    base = num_sticks(n)
    def helper(h):
        if h == 0:
            return base
        else:
            return helper(h-1) + base + (n+1)**2
    return helper

# iterative solution
def make_stick_counter(n):
    base = num_sticks(n)
    def helper(h):
        num = base
        for i in range(h):
            num += base + (n+1)**2
        return num
    return helper
```

## Question 4A

[2 marks]




## Question 4B

[4 marks]

```
def make_lens(name, a, b, c, d):
    return (name, a, b, c, d)




def name(lens):
    return lens[0]




def beam(lens, img):
    h,w = height(img),width(img)
    return make_image(h*lens[1] + w*lens[3], h*lens[2] + w*lens[4])
```

## Question 4C

[4 marks]

```
def make_projector(name):
    return (name,)




def place_lens(proj, lens):
    return proj + (lens,)
```

## Question 4D

[4 marks]

```
def get_lenses(proj):
    lenses = ()
    for l in proj[1:]:
        lenses += (name(l),)
    return lenses
```

## Question 4E

[4 marks]

```
def projection(proj, img):
    for lens in proj[1:]:
        img = beam(lens,img)
    return img
```

## Question 4F

[6 marks]

By calling `place_lens(cinema, vintage_proj)`, the entire projector is placed as a lens into cinema. When `projection` is called, it iterates through every lens, and eventually reaches the projector.

Now it will call `beam` on the projector instead of the lens. In our implementation of `beam`, it will try to access the refraction factors but instead obtain lenses from the projector. The height and width will be multiplied by the lenses, which are tuples and then concatenated together.

So no error will occur but the resulting image will be corrupted as the height and width are now tuples.

— END OF ANSWER SHEET —