CS1010S — Programming Methodology
School of Computing
National University of Singapore

Re-Midterm Test

16 October 2015 **Time allowed:** 1 hour 45 minutes

Matriculation No:	**S**	**O**	**L**	**U**	**T**	**I**	**O**	**N**	**S**

Instructions (please read carefully):

1. Write down your matriculation number on the **question paper**. DO NOT WRITE YOUR NAME ON THE QUESTION SET!
2. This is **an open-sheet test**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FOUR (4) questions** and **EIGHTEEN (18) pages**. The time allowed for solving this test is **1 hour 45 minutes**.
4. The maximum score of this test is **100 marks**. Students attempting the midterm test for the second time is subject to a **maximum score of 60 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. The back-sides of the sheets and the pages marked "scratch paper" in the question set may be used as scratch paper.
8. You are allowed to un-staple the sheets while you solve the questions. Please make sure you staple them back in the right order at the end of the test.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like (no red color, please).

GOOD LUCK!

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
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. Partial marks will be awarded for workings if the final answer is wrong.

A. [5 marks]

```
x = 15
if not x:
    print("Hello")
if True:
    print("World")
else:
    print("Bye")
```

```
World
```

B. [5 marks]

```
a = 10
b = 20
c = 30
def f(b):
    c = 5
    def g(b):
        return a + b + c
    return g(a)
print(f(c) + b)
```

```
45
```

C. [5 marks]

```
def foo(x, y):
    return (x*1 + y*2)
print(foo((1, ), (foo(2, 3), )))
```

```
(1, 8, 8)
```

D. [5 marks]

```
x, total = 1, 0
while x < 10:
    if x % 2:
        x = 0
    else:
        x += 1
    total += x
print(total)
```

Infinite loop. Because when x = 1, the next loop it will be 0, and when x = 0, the next loop will be 1. So x will always be < 10.

E. [5 marks]

```
def boo(x):
    return lambda y: y(x)
print(boo(boo(123)(str))(tuple))
```

```
('1', '2', '3')
```

F. [5 marks]

```
def g(tup):
    count = 0
    for element in tup:
        count += 1
print(g(tuple(range(5))))
```

```
None
```

Question 2: The Power of Two [24 marks]

Given a positive integer n, we can determine if n is a power of two by repeatedly dividing n by 2 until either i) it is no longer divisible by 2, in which case it means n is not a power-of-two, or ii) we end up with 1, which means n is a power-of-two.

A. Write a recursive function `power_of_two` which takes in a non-negative integer n and returns `True` if n is a power-of-two and `False` otherwise, **using the algorithm mentioned above**. [4 marks]

```
def power_of_two(n):

    if n == 1:
        return True
    elif n % 2 == 1:
        return False
    else:
        return power_of_two(n/2)
```

B. What is the order of growth in terms of time and space for the function you wrote in Part (A) in terms of n. Explain your answer. [4 marks]

Time: $O(\log n)$, where n is the input number. This is because for a number $n = 2^k$, the number of times needed to reach 1 by dividing by 2 is k. Since $\log_2 n = k$, there is $\log_2 n$ recursive calls.

Space: $O(log_n)$, where n is the input number. Each recursive call takes up space in the stack, and as explained above, there are $\log_2 n$ recursive calls.

C. Write an iterative function `power_of_two` which takes in a non-negative integer n and returns `True` if n is a power-of-two and `False` otherwise, **using the algorithm mentioned above**. [4 marks]

```
def power_of_two(n):

    while n > 1:
        if n % 2 == 1:
            return False
        else:
            n = n / 2
    return True
```

D. What is the order of growth in terms of time and space for the function you wrote in Part (C) in terms of n. Briefly explain your answer. [4 marks]

Time: $O(\log n)$ because the loop will loop until $n = 1$. Each iteration divides the number by 2, and as explained in part B, there are $\log 2n$ divisions until $n = 1$.

Space: $O(1)$ as there is no delayed operations or recursive calls. The variable n is updated in place so no extra space is required.

E. We can generalize the algorithm to find powers of other integers. To test if a number n is a power of x, we just keep dividing n by x until it is either no longer divisible by x or we reach 1.

Write a function `power_of_x` which takes in two arguments x and n, and returns `True` if n is a power of x and `False` otherwise. [4 marks]

```
def power_of_x(x, n):
    if n == 1:
        return True
    elif n % x != 0:
        return False
    else:
        return power_of_x(n, n / x)
```

F. We can also write a function that creates specific `power_of_x` functions. For instance:

```
>>> power_of_3 = create_power_of(3)
>>> power_of_3(27)
True

>>> power_of_5 = create_power_of(5)
>>> power_of_5(125)
True
```

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

```
def create_power_of(x):
    return lambda n: power_of_x(x, n)
```

Question 3: Higher-Order Function [24 marks]

Consider the following higher-order function that we call `flop`:

```
def flop(op, fn, a, nxt):
    if a <= 1:
        return fn(a)
    else:
        return op(fn(a), flop(op, fn, nxt(a), nxt))
```

A. Suppose the function `sum_integers(n)` computes the sum of integers from 1 to n (inclusive) and `sum_integers(n)` is defined as follows:

```
def sum_integers(n):
    <PRE>
    return flop(<T1>,
                <T2>,
                <T3>,
                <T4>)
```

Please provide possible implementations for the terms `T1`, `T2`, `T3` and `T4`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional <PRE>:	
<T1>:	`lambda x, y: x + y`
<T2>:	`lambda x: x`
<T3>:	`n`
<T4>:	`lambda x: x - 1`

B. Suppose the function `sum_even_integers(n)` computes the sum of even integers from 1 to n (inclusive) and `sum_even_integers(n)` is defined as follows:

```
def sum_even_integers(n):
    <PRE>
    return flop(<T5>,
                <T6>,
                <T7>,
                <T8>)
```

Please provide possible implementations for the terms `T5`, `T6`, `T7` and `T8`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional <PRE>:	
<T5>:	`lambda x, y: x + y`
<T6>:	`lambda x: x if x%2 == 0 else 0`
<T7>:	`n`
<T8>:	`lambda x: x - 1`

C. **[Warning: HARD!]** We can also express `map` (described in the appendix) in terms of `flop` as follows:

```
def map(fn, tup):
    <PRE>
    return flop(<T13>,
                <T14>,
                <T15>,
                <T16>)
```

Provide a possible implementation for the terms `T13`, `T14`, `T15` and `T16`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional `<PRE>`:	
`<T13>`:	`lambda x, y: y + x`
`<T14>`:	`lambda x: ( fn(tup[x-1]), )`
`<T15>`:	`len(tup)`
`<T16>`:	`lambda x: x - 1`

D. **[Warning: HARD!]** We can also express the `power_of_two` function from Question 2 in terms of `flop` as follows:

```
def power_of_two(n):
    <PRE>
    return flop(<T9>,
                <T10>,
                <T11>,
                <T12>)
```

Please provide possible implementations for the terms `T9`, `T10`, `T11` and `T12` without using any previously defined functions in Question 2. You may also optionally define other helper functions in `<PRE>` if needed.

Hint: The expression "`a and b`" evaluates to `True` only when both `a` and `b` are `True`. [6 marks]

*optional <PRE>:	
<T19>:	`lambda a, b: a and b`
<T10>:	`lambda x: True if x % 2 == 0 or x == 1 else False`
<T11>:	`n`
<T12>:	`lambda x: x // 2`

Question 4: Dominoes [22 marks]

Warning: Please read the entire question clearly before you attempt this problem!!

Dominoes is a game played with rectangular "domino" tiles. Each domino is a tile with a line dividing its face into two square ends. Each end is marked with a number of spots.

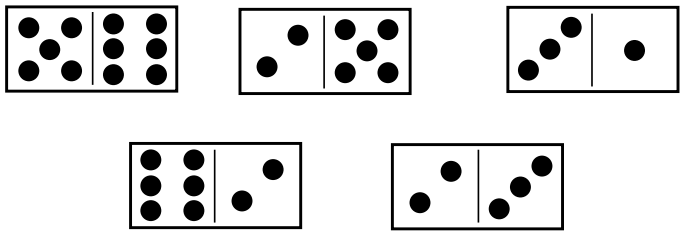


Figure 1: Some examples of dominoes.

We can write a Python function `make_domino(left, right)` which takes in two integers and creates a domino object with the numbers `left` and `right` on its face. For our purpose, we assume the dominoes cannot be rotated. For example, the dominoes in Figure 1 can be created by:

```
>>> dom56 = make_domino(5, 6)
>>> dom25 = make_domino(2, 5)
>>> dom31 = make_domino(3, 1)
>>> dom62 = make_domino(6, 2)
>>> dom23 = make_domino(2, 3)
```

The functions `left` and `right` each takes in a domino and returns the left and right number of the domino respectively. For example:

```
>>> left(dom56)
5

>>> right(dom56)
6
```

A. How can you use a tuple to represent a domino? [1 marks]

A tuple of two elements, where the first element is the left number and the second element is the right number.

B. Provide an implementation for the functions `make_domino`, `left` and `right`. [4 marks]

```
def make_domino(left, right):
    return (left, right)


def left(domino):
    return domino[0]


def right(domino):
    return domino[1]
```

A hand is a sequence of dominoes that follows a strict rule: adjacent dominoes have to touch with a matching value. Figure 2 shows an example of a valid hand.

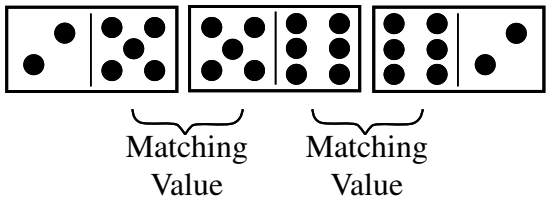


Figure 2: A valid hand.

This is valid because the first domino touches the second with the matching value 5, and the second touches the third with the matching value of 6.

A hand can be represented in Python as a tuple of dominoes and is support by the following functions:

- `empty_hand()` creates an empty hand with no dominoes.
- `add_left(domino, hand)` takes as arguments a domino and a hand, and adds the domino to the left side of the hand. It returns a new hand with the added domino if the new hand is valid, otherwise, it returns `None`.
- `add_right(domino, hand)` performs the same action as `add_left` but adds the domino to the right of the hand.
- `join(l, r)` takes as inputs two hands, one left, one right, and returns a new hand with the left joined to the right. If the new hand is not a valid hand, `None` is returned instead.

Important: For the following questions, assume that you are unaware of the exact implementation of a domino and only have access to the accessor functions defined in part B.

C. Write a series of statements that will produce the hand shown in Figure 2 into a variable `myhand`. You may use the domino variables defined at the beginning of the question. [4 marks]

```
myhand = empty_hand()
myhand = add_right(dom25, myhand)
myhand = add_right(dom56, myhand)
myhand = add_right(dom62, myhand)
```

D. Provide an implementation for the functions `empty_hand`, `add_left` and `add_right`. [4 marks]

```
def empty_hand():
    return ()



def add_left(domino, hand):
    if hand == ():
        return (domino,)
    elif right(domino) == left(hand[0]):
        return (domino,) + hand
    else:
        return None



def add_right(domino, hand):
    if hand == ():
return (domino,)
elif left(domino) == right(hand[-1]):
return hand + (domino,)
else:
return None
```

E. Provide an implementation of `join`. [3 marks]

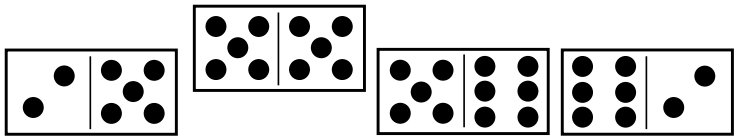
```
def join(l, r):
    if right(l[-1]) == left(r[0]):
        return l + r
    else:
        return None
```

F. The value of a hand is simply the sum total of the numerical values on the faces of the dominoes. For example, the value of the hand shown in Figure 2 is $2+5+5+6+6+2=26$.

Implement the function `value_of(hand)` that takes in a hand and returns the value of the hand. You may wish to use the functions `map`, `filter` or `accumulate` given in the Appendix.[3 marks]

```
def value_of(hand):
    return accumulate(lambda x,y: x+y,
                      0,
                      map(lambda d: left(d) + right(d),
                          hand))
```

G. The function `insert_between(domino, hand, pos)` takes a domino and inserts it into the hand, at the position given by `pos`, that is, after a successful insert operation, `hand[pos] == domino`.



The above figure shows an example where a (5, 5) domino is inserted into the hand in Figure 2 at position 1.

The function `insert_between(domino, hand, pos)` returns a new hand if the insertion is valid, and returns `None` if it results in an invalid hand.

Provide an implementation of the function `insert_between(domino, hand, pos)`. [3 marks]

```
def insert_between(domino, hand, pos):
    if pos == 0:
        # same as adding to the left
        return add_left(domino, hand)
    elif pos == len(hand):
        # same as adding to the right
        return add_right(domino, hand)
    else:
        # split the hand and add to the right
        new = add_right(domino, hand[:pos])
        if new == None:  # return None if the add is invalid
            return None
        # then join the other half of the hand
        return join(new, hand[pos:])
```

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

— E N D O F P A P E R —