CS1010X — Programming Methodology
School of Computing
National University of Singapore

Re-Midterm Test

20 April 2019 **Time allowed:** 1 hour 45 minutes

Student 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 **FIVE (5) 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**. 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		
Q5		
Total		

Question 1: Python Expressions [24 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, explain why. Partial marks may be awarded for workings if the final answer is wrong.

A.

```
x, y = 999, 11
def Z(x, y):
    def W(x):
        if x%10 < y:
            print("ok!")
        else:
            print("not ok!")
        return x/10
    return W(y)
y = Z(y, x)
print(str(y) + "% sure!")
```

[4 marks]

```
ok!
99.9% sure!
```

Warm up: Tracing - a taste of scoping too.

B.

```
num = (9,8,7,6,)
j=0
for i in num:
    j += 1
print (j + (num in (num)))
```

[4 marks]

```
4
```

Warm up further....

C.

```
def bar(a,b):
    if b%a >= b-a:
        print(a, "first")
    elif b*3 > a**2 and "False":
        print(a, "second")
    else:
        print(a, "third")
    a,b = b%a,a
    if a>1:
        bar(a,b)
bar(7,19)
```

[4 marks]

```
7 second
5 first
2 second
```

Testing code tracing for if-then-else conditions and also that students understand `and`.

D.

```
def perfect():
    cool = ()
    for i in (1,2,4,5,7,9,11,12):
        for j in range(i):
            if i*j>20:
                if j%2==0:
                    cool += (i,)
                break
    return cool
print("perfect",perfect())
```

[4 marks]

```
perfect (11, 12)
```

Test that students understand iteration, `range` and `break`.

E.

```
def generator(seed):
    def gen(bag, seed):
        if (seed == ""):
            print(bag)
        else:
            i = 0
            for s in seed:
                gen(bag+s, seed[0:i] + seed[i+1:])
                i = i + 1
    gen("", seed)
generator("123")
```

[4 marks]

```
123
132
213
231
312
321
```

Test of recursion and slicing

F.

```
z = lambda x: x**3
y = lambda y: z
x = lambda x: y(z)
z = y(x)
print(x(2)(4))
```

[4 marks]

```
64
```

Test on handling a function that returns a function.

Question 2: Number Patterns [21 marks]

Consider the following series of integers:

$$1, 12, 123, 1234, \cdots, 123456789, 1234567890, 12345678901, \cdots$$

Basically, we add another digit in each step in increasing sequence and wrap around after 0. We define this series as $f(n)$, such that $f(1) = 1$, $f(2) = 12$, and so on.

A. **[Warm Up]** Provide an implementation of f for an input n. [3 marks]

```
def f(n):
    if n==1:
        return 1
    else:
        return f(n-1)*10 + n%10

def f(n):
    total=0
    for i in range(1,n+1):
        total = 10*total + i%10
    return total
```

Next, consider the following series of integers:

$$1, 112, 112123, 1121231234, \cdots$$

Basically, the series is created by concatenating the digits of f. We define this series as $g(n)$, such that $g(1) = 1$, $g(2) = 112$, and so on.

B. Provide an <u>recursive</u> implementation of g with an input n. [4 marks]

```
def g(n):
    if n==1:
        return 1
    else:
        return g(n-1)*(10**n) + f(n)
```

C. What is the order of growth in time and in space for the function you wrote in Part B (in terms of n)? If you referenced f in your solution, assume the implementation of f in Part A. [2 marks]

Time: $O(n^2)$

Space: $O(n^2)$

This answer assumes a recursive implementation of f with $O(n)$ time and $O(n)$ space.

D. Provide an <u>iterative</u> implementation of g with an input n. [4 marks]

```
def g(n):
    total = 0
    for i in range(1,n+1):
        for j in range(i):
            total *= 10
        total += f(i)
    return total
```

E. What is the order of growth in time and in space for the function you wrote in Part D (in terms of n)? Again, if you referenced f in your solution, assume the implementation of f in Part A. [2 marks]

Time: $O(n^2)$

Space: $O(n)$

This answer assumes a recursive implementation of f with $O(n)$ time and $O(n)$ space.

Finally, consider the following palindromic series of integers:

$$1, 121, 12321, \cdots, 12345678987654321, 1234567890987654321, 123456789010987654321, \cdots$$

We define this series as $h(n)$, such that $h(1) = 1$, $h(2) = 121$, and so on.

F. Provide an implementation of h with an input n, **subject to the constraint that you cannot use the `str` function**. [4 marks]

```
def h(n):
    result = 0
    for i in range(1, n):
        result = result*10 + i%10
    for j in range(n, 0, -1):
        result = result*10 + j%10
    return result
```

G. What is the order of growth in time and in space for the function you wrote in Part F (in terms of n). [2 marks]

Time: $O(n)$

Space: $O(1)$

Question 3: Higher Order Functions [21 marks]

Consider the following higher order functions:

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

def foo(op, a, f, terminate, next, n):
    if terminate(a,n):
        return f(a)
    else:
        return op(f(n), foo(op, a, f, terminate, next, next(n)))
```

A. **[Warm Up]** Clearly we can define `fold` in terms of `foo` as follows:

```
def fold(op, f, n):
    return foo(op, 0, f, <T1>, <T2>, n)
```

Please provide possible implementations for T1 and T2. [4 marks]

<T1>: [2 marks]	`lambda a,n: n==0`
<T2>: [2 marks]	`lambda x: x-1`

B. **[Warm up a bit more]** `odd_sum(n)` returns the sum of the first n odd positive integers, i.e.

```
>>> odd_sum(1)   % 1
1
>>> odd_sum(2)   % 1+3
4
>>> odd_sum(3)   % 1+3+5
9
```

We can define `odd_sum(n)` in terms of `fold` as follows:

```
def odd_sum(n):
    return fold(<T3>, <T4>, <T5>)
```

Please provide possible implementations for T3, T4 and T5. [5 marks]

<T3>: [1 marks]	`lambda x,y: x+y`

<T4>: [2 marks]	`lambda x: 0 if x%2==0 else x`
<T5>: [2 marks]	`2*n`

C. The function `factorial` takes a positive integer n and returns the product of the first n positive integers, i.e. `factorial(n) = n!`. Clearly we can define `factorial` in terms of `foo` as follows:

```
def factorial(n):
    return foo(lambda x,y: x*y, <T6>, <T7>, <T8>, <T9>, n)
```

Please provide possible implementations for T6, T7, T8, and T9. [6 marks]

<T6>: [2 marks]	`1`
<T7>: [2 marks]	`lambda x: x`
<T8>: [1 marks]	`lambda a,n: n==0`
<T9>: [1 marks]	`lambda x: x-1`

D. Interestingly, we can also define `factorial` in terms of `foo` as follows:

```
def factorial(n):
    return foo(lambda x,y: x*y, <T10>, <T11>, <T12>, <T13>, 0)
```

Please provide possible implementations for T10, T11, T12, and T13. [6 marks]

`<T10>`: [2 marks]	`n-1`
`<T11>`: [2 marks]	`lambda x: x+1`
`<T12>`: [1 marks]	`lambda a,n: a==n`
`<T13>`: [1 marks]	`lambda x: x+1`

Question 4: Text Editor Undo [31 marks]

Given your newfound Python programming skills, you managed to find an internship with a reputable software company, called Macrohard, that is in the midst of developing a new flagship this text-processing software Macrohard WordStar. Your new boss has asked you to build a new `word` object for this new software. The new object would help to track the edits made to a word (string) in case the user decides to undo his edits.

You are required to implement the following 8 functions in the problem to support this `word` object:

1. `make_word` takes in a string and returns a new word object that represents the input string.
2. `is_word` takes an object and returns `True` if it is a valid word object, or `False` otherwise.
3. `get_word` takes a word object returns the string represented by the object.
4. `delete` takes a word object and an integer index returns a new word object with the letter (one character string) at the specified index deleted. If the index is not valid, the original word object is returned and this function has no effect.
5. `insert` takes a word object, an integer index and a letter (string) and returns a new word object with the letter inserted at the specified index. The index should be from 0 to the length of the string represented by the word object. If the index is not valid, the original word object is returned and this function has no effect.
6. `undo` takes a word object and returns a new word object with the most recent `insert` or `delete` reversed. If there has not been any operations on the word object, then the original word object is returned and this function has no effect.
7. `accept_all_changes` takes a word object and returns a new word object that has no history of operations, i.e. not possible to undo. If there has not been any operations on the word object, then the original word object is returned and this function has no effect.
8. `equal` takes 2 word objects and returns `True` if they represent the same string, or `False` otherwise.

You are advised to read through all the requirements for this question carefully before deciding on the implementation for your word object.

Example execution:

```
>>> w1 = make_word("blue")
>>> get_word(w1)
'blue'
>>> is_word(w1)
True
>>> is_word("blue")
False

>>> w2 = delete(delete(w1,2),2)
>>> get_word(w2)
'bl'

>>> w3 = insert(insert(insert(w2,2,"k"),2,"c"),2,"a")
>>> get_word(w3)
'black'
>>> get_word(undo(w3))
'blck'
>>> get_word(undo(undo(w3)))
'blk'
>>> get_word(undo(undo(undo(w3))))
'bl'
>>> get_word(undo(undo(undo(undo(w3)))))
'ble'
>>> get_word(undo(undo(undo(undo(undo(w3))))))
'blue'

>>> w4 = make_word("black")
>>> equal(w3,w4)
True
>>> equal(w3,w2)
False

>>> w5 = accept_all_changes(w3)
>>> equal(w4,w5)
True
>>> get_word(w5)
'black'
>>> get_word(undo(w5))
'black'
```

A. Decide on an implementation for the word object and implement `make_word`. Describe how the state is stored in your implementation and explain how you will track changes to the word object. [5 marks]

Note: You are limited to using **tuples** for this question, i.e. you cannot use lists and other Python data structures.

There are many possibilities. Here, we store different versions of the words by appending to a tuple.

```
def make_word(string):
    return ("#word", string)
```

`"#word"` is the label for the word object. As the string is edited, the new strings are appended to the end of the tuple representation. `obj[-1]` is the string represented by the object `obj`.

If the design is just simply a "normal" tuple that keeps, for example, string after string, then this is not a good design as it does not differentiate such a tuple from an Word object. Full credit cannot be awarded as such.

B. Implement the function `is_word(obj)` that returns `True` if `obj` is a valid word object, or `False` otherwise. [4 marks]

```
def is_word(obj):
    return type(obj) == tuple and len(obj) >=2 and obj[0] == "#word"
```

C. Implement the function `get_word` that returns the string represented by a word object. [2 marks]

```
def get_word(word):
    return word[-1]
```

D. Implement the function `delete(word, index)` that returns a new word object with the letter at the specified index deleted. If the index is not valid, the original word object is returned and this function has no effect. No effect means that this operation will not be an operation subject to undo. [3 marks]

```
def delete(word, index):
    if index < 0 or index > len(word[-1])-1:
        return word
    else:
        original = word[-1]
        return word + (original[:index]+original[index+1:],)
```

E. Implement the function `insert(word, index, letter)` that returns a new word object with the specified letter inserted at the specified index. The index should be from 0 to the length of the string represented by the word object. If the index is not valid, the original word object is returned and this function has no effect. No effect means that this operation will not be an operation subject to undo. [3 marks]

```
def insert(word, index, letter):
    if index < 0 or index > len(word[-1]):
        return word
    else:
        original = word[-1]
        return word + (original[:index]+letter+original[index:],)
```

F. Implement the function `undo(word)` that returns a new word object with the most recent `insert` or `delete` reversed. If there has not been any operations on the word object, then the original word object is returned and this function has no effect, since there is nothing to undo to begin with. [3 marks]

```
def undo(word):
    if len(word)==2:
        return word
    else:
        return word[:-1]
```

G. Implement the function `accept_all_changes(word)` that returns a new word object that has no history of operations, i.e. not possible to undo. If there has not been any operations on the word object, then the original word object is returned and this function has no effect. [3 marks]

```
def accept_all_changes(word):
    if len(word)==2:
        return word
    else:
        return (word[0],)+(word[-1],)
```

H. Implement the function `equal(word1, word2)` that takes 2 word objects and returns `True` if they represent the same string, or `False` otherwise. [3 marks]

```
def equal(word1, word2):
    return word1[-1] == word2[-1]
```

I. Suppose instead of keeping track of all changes, the word object should only track up to the last 3 changes at any point. Which of the functions you implemented above would have to be changed to support this requirement? Describe exactly what you need to do. [5 marks]

Example execution:

```
>>> w1 = make_word("blue")
>>> w2 = delete(delete(w1,2),2)
>>> w3 = insert(insert(insert(w2,2,"k"),2,"c"),2,"a")
>>> get_word(undo(w3))
'blck'
>>> get_word(undo(undo(w3)))
'blk'
>>> get_word(undo(undo(undo(w3))))
'bl'
>>> get_word(undo(undo(undo(undo(w3)))))
'bl'
>>> get_word(undo(undo(undo(undo(undo(w3))))))
'bl'
```

We will need to change just the `insert` and `delete` functions by limiting the number of undos tracked to a maximum of three as follows:

```
def delete(word, index):
    if index < 0 or index > len(word[-1])-1:
        return word
    else:
        original = word[-1]
        return limit(word + (original[:index]+original[index+1:],))

def insert(word, index, letter):
    if index < 0 or index > len(word[-1]):
        return word
    else:
        original = word[-1]
        return limit(word + (original[:index]+letter+original[index:],))

def limit(word):
    if len(word) > 4:
        return word[:1] + word[len(word)-4:]
    else:
        return word
```

Question 5: What did you did you learn since last midterm? [3 marks]

If you are taking this exam as a re-exam, tell us what 3 things you learnt from the midterms that has helped you do better this time. If you are taking this exam as your first midterm, tell us what you think are the 3 most important concepts you have learnt in CS1010X thus far – or you can tell us 3 things you learnt from taking this exam.

Student will get points for any reasonably attempt that demonstrates effort. Roughly 1 point is given for each "reasonable" point made.

Scratch Paper

— END OF PAPER —