SCHOOL OF COMPUTING

ASSESSMENT FOR
Special Term I, 2022/2023

Solutions for CS1010X — PROGRAMMING METHODOLOGY

June 2023 Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **7 questions** and comprises **13 pages**.
3. Weightage of each question is given in square bracket. Total score is 48.
4. This is an **OPEN** book assessment, but you are NOT allowed to use any electronic devices such as laptop, mobile phone, and calculator.
5. Write all your answers in the space (enclosed in a box after each part/question) provided in this booklet.
6. You should make effort to write your answer legibly and neatly to avoid any confusion in interpretation. Use pencil where applicable (to present your Python codes with proper indentation) to ease any amendment along the way.

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

(this portion is for the examiner's use only)

Question	Marks
Q1	
Q2	
Q3	
Q4	
Q5	
Q6	
Q7	
Total	

Question 1: Warm up - fill in the blank [2 marks]

It is a belief that the more codes that we write, the more ___ **mistake** ___ we can make.

We are thus better off with _____ **reusing** _____ other people's tested codes.

Question 2: Python Expressions [13 marks]

There are several parts to this problem. Answer each part **independently and separately**. It is assumed that there is no syntax errors in our codes.

A. [4 marks]

```
a = 1
def test(b):
    d = a
    def t1(c):
        c = a + b + c + d
        return c
    return t1
z = test(4)
print(z(2))
a = 11
print (z(11))
```

```
8
27
```

B. [4 marks]

```
lst = []
t = (1,2,3)
lst.append(t)
t = t + ()
print(lst)
lst.append(t)
print(lst)
lst.extend(t)
lst[1]="b"
print(lst)
print(t)
```

```
[(1, 2, 3)]
[(1, 2, 3), (1, 2, 3)]
[(1, 2, 3), 'b', 1, 2, 3]
(1, 2, 3)
```

C.

```
def sum(t, a, b):
    if a > b or a==b:
        return 0
    else:
        return t[a] + sum(t, a+1, b)

def filter(f, t, a, b):
    if a > b or a==b:
        return ()
    else:
        if f(t[a]) == True:
            return (t[a],) + filter(f, t, a+1, b)
        else:
            return filter(f, t, a+1, b)

def map(f, t):
    def helper(f, t, a, b):
        if a > b or a==b:
            return ()
        else:
            return (f(t[a]),) + helper(f, t, a+1, b)
    return helper(f, t, 0, len(t))

t = (1,6,7,5,4,8,2,3)
print( map( lambda x:x**2,
            filter(lambda x: x > sum(t, 0, len(t))/len(t),
                   t, 0, len(t))))
```

[3 marks]

```
(36, 49, 25, 64)
```

```
# continue from above with a "larger" t, but using the same print statement

t = tuple(range(1,101)) # t = (1, 2, 3, 4, 5,..., 97, 98, 99, 100)
print( map( lambda x:x**2,
            filter(lambda x: x > sum(t, 0, len(t))/len(t),
                   t, 0, len(t))))

# You can provide (in words or in formula, whichever is convenient)
# the output from the print statement.
# This is to test your understanding of the semantic of the print statement.
```

[2 marks]

$(51^2, 52^2, ..., 98^2, 99^2, 100^2)$

The function is to take the square of each number that is larger than the mean of all the numbers.

Question 3: Iteration & Recursion [10 marks]

Suppose you are given the following:

```
def double_fold(op, f, a, b, c):
  if a == b:
      return f(a)
  elif a > b:
      return c
  else:
      return op( f(a), double_fold(op, f, a+1, b-1, c), f(b))
```

A. [2 marks]

State the output of the following statement:

```
print( double_fold( lambda x, y, z: x + y + z,
                    lambda x: x**2,
                    3, 6, 0) )
```

86

Note: This part is there to help you understand the execution of the function so as to prepare you for the last part.

B. [2 marks]

State the output of the following statement:

```
print( double_fold( lambda x, y, z: x + y - z,
                    lambda x: x**2,
                    3, 7, 1) )
```

-35

Note: This part is there to help you understand the execution of the function so as to prepare you for the last part.

C. [4 marks]

Provide an iterative implementation of the given function that produces the same output value when called with the same set of inputs. The codes needed for the simple cases (when `a` $\geq$ `b`) have been provided to you. Your task is to complete the rest of the function (when `a` $<$ `b`). Do provide comments to explain your work to still receive partial credit for your effort.

```
def double_fold_itr(op, f, a, b, c):
  if a==b:
      return f(a)
  elif a > b:
      return c

  # start you codes below when a < b

  # 2 marks for handling even, and 2 marks for handling odd
  even = ( (b - a + 1)%2 == 0 )
  if even:
      result=c
      i=a+(b-a)//2
      j=i+1
  else: # odd
      result=f(a+(b-a)//2)
      i=a+(b-a)//2 - 1
      j=i+2

  while i >= a:
      result = op( f(i), result, f(j) )
      i -= 1
      j += 1
  return result
```

D. [2 marks]

State and explain whether your above implementation `double_fold_itr` has a different time complexity and space complexity from the given function `double_fold` when the inputs are the same. We assume that both `f` and `op` take constant time. No credit will be awarded if your `double_fold_itr` is far from correct.

There is no change in the time complexity as the above iterative implementation still repeatedly calculates some same values in the loop. But, the space complexity can be constant (when f(n) is constant) as there is no longer any deferred operation.

Question 4: Polynomial [7 marks]

This question is about multiple representations. You are NOT allowed to use any library packages, such as NumPy, to solve our questions here. Your working should be based on list (Part A) and dictionary (Part B) as given in Lecture 8.

For a polynomial:

$$P = 3x^{10} + 5x^8 + 2x^3 + 7$$

we say each integer in front of x^i as the coefficient of x^i. For example, 3 is the coefficient of x^{10}, and 7 is the coefficient of x^0 (note that x^0 is just 1). The differentiation of our example polynomial P is

$$Q = 30x^9 + 40x^7 + 6x^2$$

That is, the differentiation is to take, one by one, each term ax^b (such as $3x^{10}$ in our P, where $a = 3, b = 10$) to produce $(a \times b)x^{b-1}$ (such as $30x^9$ for our example). For a term that is a constant c (such as 7 in our P), it is actually cx^0 as mentioned, and thus its differentiation is 0.

A. For this part, our polynomial is represented with a list. So, P is represented as $[3,0,5,0,0,0,0,2,0,0,7]$ where the last term in the list is the constant, and then the second last term is the coefficient of x^1, and then the third last term is the coefficient of x^2 and so on. Then Q, as the differentiation of P, is represented in a list as $[30,0,40,0,0,0,0,6,0,0]$. You write a function `differentiate_lst` with an input polynomial (in list) to output its differentiation (in list). [3 marks]

```
def differentiate_lst (t):
    if t==[]:
        return []
    s=[]
    for i in range(len(t) - 1):
        s.append( (len(t)-1-i) * t[i] )
    return s
```

B. For this part, our polynomial is represented with a dictionary. So, P is represented as {10:3, 8:5, 3:2, 0:7} where each term b:a (such as 10:3) represents the ax^b (that is $3x^{10}$ for our example of 10:3) in the polynomial. Then Q, as the differentiation of P, is represented in a dictionary as {9:30, 7:40, 2:6}. You write a function `differentiate_dic` with an input polynomial (in dictionary) to output its differentiation (in dictionary). [3 marks]

```
def differentiate_dic (t):
    if t=={}:
        return {}
    s={}
    for x,y in t.items():
        if x > 0:
            s[x-1] = x*y
    return s
```

C. Does this remind you anything about the various matrix representations? Provide a few sentences on your thoughts. [1 marks]

Dictionary is like the sparse matrix representation, whereas list is the normal representation.

Question 5: Dynamic Programming Revisit, once again [7 marks]

We now revisit the cut rod problem (in Recitation #10). To recap, you have a rod of length n meters and you hope to cut it into pieces and sell them with maximum profit. You can only cut pieces into integer lengths. The profit of rod with different lengths are as shown in the `prices` variable where, in our below example, for a piece of length 1 we can sell it for 1 dollar, length 2 for 5 dollars, length 3 for 8 dollars and so on.

```
def cut_rod(n, prices):
  max_price = [0] * (n+1)
  for length in range(1, n+1):
      for p in prices:
          if p <= length:
              max_price[length] = max(max_price[length], \
                                      prices[p] + max_price[length-p])
  return max_price[n]
prices = {1:1, 2:5, 3:8, 4:9, 5:10, 6:17, 7:17, 8:20, 9:24, 10:30}
print( cut_rod(4, prices) )   # output is 10 dollars
>>> 10
```

A. Besides knowing the profit, we now want to know the cutting needed to achieve the maximum profit too. For example, if we do

```
print( cut_rod_t (4, prices) )
>>> (10, (2, 2))
# where (2, 2) means the rod is cut into two pieces of length 2 each
# with total profit of 10 dollars

print( cut_rod_t (29, prices) )
>>> (85, (3, 6, 10, 10))
# where (3, 6, 10, 10) means the rod is cut into four pieces of
# lengths 3, 6, 10, 10 with total profit of 85
```

You are to modify the above `cut_rod` to become `cut_rod_t` as discussed. Please note that you are to maintain the same given code structure for your answer. [3 marks]

```
def cut_rod_t (n, prices):
    max_price = [0] * (n+1)
    soln = [ () ] * (n+1)
    for length in range(1, n+1):
        for p in prices:
            if p <= length:
                if max_price[length] < (prices[p] + max_price[length-p]):
                    max_price[length] = prices[p] + max_price[length-p]
                    soln[length] = (p,) + soln[length-p]
    return max_price[n], soln[n]
```

B. For your solution, state the space complexity in terms of `n` and provide your reasoning. We assume the length of `prices` is some constant. [2 marks]

The space complexity is $O(n^2)$ as it needs to have `soln` and `max_price` of length `n`, but the tuple inside each `soln[`ℓ`]` can be of length `n`.

C. For your solution, state the time complexity in terms of `n` and provide your reasoning. We assume the length of `prices` is some constant. [2 marks]

The time complexity is $O(n^2)$ as it needs to fill in `soln` from 1 till `n`. Each time it has to go through each `p` in `prices` of a constant length m. For a particular $\ell \in \{1, m\}$, there are at most m changes to `soln(`ℓ`)` but each `soln(`ℓ`)` can have length `n`. So, total time is the same as that for the space.

Question 6: Java Exercise [4 marks]

Study the following classes and write the output (in 4 lines) from the main routine.

```
class A {
    // variable "a" below is declared as "static"
    // this means there is only 1 copy of "a" shared by ALL instances of A
    protected static int a = 5;
    public A ( )     { this(2);  }   // passing 2 to the constructor with one parameter, i
    public A (int i) { a = a + i; }  // this constructor has one parameter, i
    public int getA( ) { return a; }
}

class B extends A {
    protected int b = 0 ;
    public B ( ) { this(6); }   // passing 6 to the constructor with one parameter, k
    public B (int k) {          // this constructor has one parameter, k
        if (a > k)  b = k;
        else        b++;
    }
    public int getB( ) { return b; }
}

class D extends B {
    private int d;
    public D (int m) {
      super(9);
      d = m;
    }
    public int getDplus ( ) { return d + a;}
}

class Q {
    public static void main (String[ ] args) {
        B binstance = new B();
        System.out.println (binstance.getB() );
        System.out.println (binstance.getA());
        D dinstance = new D(0);
        System.out.println (dinstance.getB() );
        System.out.println (dinstance.getDplus() );
    }
}
```

```
6
7
1
9
```

Question 7: Computational Thinking [5 marks]

Study the following codes carefully. We shall assume that there is no numerical error even though we use floating point number in our calculation. For example, the decrement of `rate` by 0.2 in the loop will result in precising 1.8 (not, for example, 1.79999999999998) after the first decrement, and then 1.6 (not, for example, 1.59999999999998) after the second, etc. Same for the increment of `x` by 0.2.

```
def strange(n):
    rate = 2
    x = 1
    value = 0
    counter = 0
    while value < n:
        counter += 1
        value = x*rate
        if value < n:
            value = 0
            oldRate = rate
            rate = rate - 0.2
            if rate <= 1.5:
                rate = oldRate
            else:
                x = 1
        x += 0.2
    return counter
```

A. Give the output of the following `print` statement. [2 marks]

```
print( strange(3) )
```

7
Note: notice that rate will stay at 1.6 once it is there, and `x` will keep increasing by 0.2 each time after that. Use of floating point number here is dangerous but we want to do this exercise with what you know from integer numbers.

With the above, we want to understand the time and space complexity of the codes with respect to the input `n`, which is always a positive integer.

B. What is the space requirement of the codes with respect to input `n`? State your answer first, and then provide your reasoning. [1 marks]

O(1) as the codes use only a constant number of variables.

C. What is the time requirement of the codes with respect to input `n`? State your answer first, and then provide your reasoning. [2 marks]

The time taken is recorded by the variable `counter`. The value of this `counter` is linearly proportional to input `n`.

The value of the `rate` stays at 1.6 after 3 loopings, and `x` keeps increasing by 0.2 in each loop after that. So, the value calculated from `x*rate` keeps increasing by 0.32 in each loop until it is larger than `n`. So, the number of loopings needed is just 3 to 4 times for each increase of 1 by `value`. Thus, we need at most 4`n` times for `value` to be no longer smaller than `n` to terminate the while-loop. So, the number of loopings is linearly proportional to `n` (with the constant of at most between 3 to 4).

— E N D O F P A P E R —