CS1010X — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

4 April 2020 **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 **FOUR (4) questions** and **SEVENTEEN (17) pages**. The time allowed for solving this test is **1 hour 45 minutes**.
4. The maximum score of this test is **30 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 [8 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**.

It is assumed that there is no syntax errors in our input to the Python interpreter. If the interpreter produces an execution error message at some stage, state where and the nature of the error (you don't need to show exactly the error message). Please note that you are still needed to write out any output from the Python expressions before the error was encountered. In another extreme, if the interpreter enters an infinite loop, explain how it happens.

Partial marks may be awarded for working (within the provided answer box) if the final answer is wrong. You are, however, responsible to make clear the part of your answer verse the part of your working.

A.

```
x = 0
y = 1
def f(y):
    if y > x:
        print("y > x")
    elif x >= y:
        print("x >= y")
    y = y + 1
    return (lambda y: "not ok" if y==False else "ok")

y = f(x)
print(y(None))
```

[2 marks]

Answer:

```
x >= y
ok
```

Warm up: Tracing - a taste of scoping too.

B.

```
def foo(y):
    def goo(z):
        if z <= 1:
            return "ok"
        else:
            z = z // 3
            print(z)
            print(goo(z))
            return goo(z//2)
    return goo(y)

print(foo(9))
```

[2 marks]

Answer:

```
3
1
ok
ok
ok
```

Testing of condition - hope many have 3 oks....

C.

```
def mylove(t):
    length = len(t)
    i = 1
    while i < length:
        s = t
        t = t[0:i]
        for j in range(0, length - i):
            t = t + (s[j] + s[j+i],)
        i *= 2
    return t

print(mylove((1, 2, 3, 4, 5, 6, 7, 8, 9, 10)))
```

[2 marks]

Answer:

```
(1, 3, 6, 10, 15, 21, 28, 36, 45, 55)
```

This is parallel prefix - i like this for this can be done in parallel very smartly.

D.

```
def blur(x):
    return lambda x: x**2
def king(x):
    return lambda x: x+2

print( blur(king(king(king)))(4) )
print( king(blur(blur(blur)))(4) )
print( blur(king(blur(king)))(4) )
print( king(blur(king(blur)))(4) )
```

[2 marks]

Answer:

```
16
6
16
6
```

Question 2: Iteration & Recursion, with Time & Space [8 marks]

Suppose you are given the following functions:

```
from math import *

def isPower_of_2(n):
    return (log(n,2) - int(log(n, 2))) == 0

def fun(n):
    if n==0:
        return 0
    elif isPower_of_2(n):
        return n + fun(n//2)
    else:
        return n + fun(n-1)
```

A. Assume `isPower_of_2` takes $O(1)$ time and $O(1)$ space. Provide the time and space complexity of the function `fun` in terms of n. Justify your answer. You may draw out an execution tree on some example value of n to show your good understanding of the problem to secure (partial) credit. [4 marks]

Time: $O(n)$

Space: $O(n)$

Justification (for time and for space):

There are two parts to this function.

The function will reduce (recursively) its value of n by 1 till n is a power of 2. This process will reach one such value by less than $n/2$ recursive calls. This part does take $O(n/2) = O(n)$ time and space.

Once n is a power of 2, it will continue to be power of 2 in the following recursive calls till it reaches 0. So, this second part is $O(\log n)$ in time and in space. So, to sum up, we have $O(n) + O(\log n)$, thus the answer is $O(n)$.

Correct answer of $O(n)$ time is 1 mark and $O(n)$ space is another 1 mark. Then good explanation is 2 marks.

B. Provide an iterative implementation of `fun` with an input n. You can continue to use the function `isPower_of_2` freely without redefining it in your answer. [2 marks]

```
def fun_itr(n):

    if n==0:
        return 0
    result = 0
    while (n >= 1):
        result += n
        if isPower_of_2(n):
            n = n//2
        else:
            n = n - 1
    return result
```

C. Now suppose we are given the following functions `fact` and `fun2`. What is the time complexity of `fun2`? Justify your answer: we can accept good approximation without you having to work out the exact formula. [2 marks]

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)

def fun2(n):
    result = 0
    for i in range(n):
        for j in range(1, n):
            if (j > i):
                result += fact(j)
    return result
```

Time: $O(n^3)$

Justification:

The calling of `fact(j)` happens j times for j from 1 to $n-1$. That is, `fact(1)` is done just 1 time (when $i = 0, j = 1$), `fact(2)` is done twice (when $i = 0, j = 2$ and $i = 1, j = 2$), etc. till `fact(n-1)` is done $(n-1)$ times (for i from 0 till $n-2$ and $j = n-1$). So, the justification for the time complexity is: $1^2 + 2^2 + 3^2 + ...(n-1)^2$, and we know (from the web) that this sum is $(n-1) \times n \times (2n-1)/6$.

see https://www.youtube.com/watch?v=OpA7oNmHobM if you are interested in the derivation.

Alternatively, you can do an approximation to see that the sum is no larger than $(n-1) \times (n-1)^2$ because each term is smaller or equal to $(n-1)^2$, and the sum is no smaller than $(n/2) \times (n/2)^2$ because each term starting from the middle one is larger than $(n/2)^2$. So, the sum is bounded above and below by $O(n^3)$.

Question 3: Higher Order Functions [6 marks]

You will be working with the following higher-order function:

```
def new_sum( t, term, next ):
    if len(t) < 1:
        return 0
    else:
        return term(t) + new_sum( next(t), term, next )
```

A. The function `even_sum` takes an input tuple τ (of numbers) to compute the sum of the even-indexed terms:

$$\texttt{even\_sum}(\tau) = \tau(0) + \tau(2) + \ldots + \tau(k)$$

where k is either the last term when the length of τ is odd or the second last term when the length of τ is even.

Example execution:

```
>>> even_sum( (1,2,3,4,5) )   # 1+3+5 = 9
9

>>> even_sum( (1,2,3,4,5,6) )   # 1+3+5 = 9
9
```

We can define `even_sum` with `new_sum` as follows:

```
def even_sum( t ):
    return new_sum( t, <T1>, <T2> )
```

Please provide possible implementations for T1 and T2. [2 marks]

<T1>: [1 marks]

```
lambda t: t[0]
```

<T2>: [1 marks]

```
lambda t: t[2:]
```

B. The function `folded_sum_square` takes an input tuple τ (of numbers) to compute the sum of the square of: the first with the last term, the second with the second last term, and so on:

$$\texttt{folded\_sum\_square}(\tau) = (\tau(0) + \tau(n-1))^2 + \ldots + (\tau(\lfloor (n-1)/2 \rfloor) + \tau(\lceil (n-1)/2 \rceil))^2$$

where n is the length of τ.

Example execution:

```
>>> folded_sum_square( (1,2,3,4,5) )  # (1+5)(1+5)+(2+4)(2+4)+(3+3)(3+3) =
108

>>> folded_sum_square( (1,2,3,4,5,6) )  # (1+6)(1+6)+(2+5)(2+5)+(3+4)(3+4) =
147
```

We can define `folded_sum_square` with `new_sum` as follows:

```
def folded_sum_square(t):
    return new_sum( t, <T3>, <T4> )
```

Please provide possible implementations for T3 and T4. [2 marks]

<T3>: [1 marks]

```
lambda t: (t[0]+t[-1])**2
```

<T4>: [1 marks]

```
lambda t: t[1:-1]
```

C. The function `alternate_sum_12` takes an input tuple τ (of numbers) to compute the sum of the terms with even term multiply by 1 and odd term multiply by 2:

$$\texttt{alternate\_sum\_12}(\tau) = \tau(0)*1+\tau(1)*2+\tau(2)*1+\tau(3)*2+\ldots+\tau(n-1)*k$$

where n is the length of τ, and k is 1 when n is odd, and k is 2 when n is even.

Example execution:

```
>>> alternate_sum_12( (1,2,3,4,5) )  # 1*1 + 2*2 + 3*1 + 4*2 + 5*1 =
21

>>> alternate_sum_12( (1,2,3,4,5,6) )  # 1*1 + 2*2 + 3*1 + 4*2 + 5*1 + 6*2 =
33
```

We can define `alternate_sum_12` with `new_sum` as follows:

```
def alternate_sum_12(t):
    return new_sum( t, <T5>, <T6> )
```

Please provide possible implementations for T5 and T6. [2 marks]

<T5>: [1 marks]

```
lambda t: t[0] if len(t) < 2 else (t[0] + t[1]*2)
```

<T6>: [1 marks]

```
lambda t: t[2:]
```

Question 4: COVID-19 [8 marks]

The current COVID-19 situation is troubling. Let's see how we can apply what we have learned to monitor it. We will do a very simple version 0.1 here, and leave any complication to another time. You can use ONLY the tuple data structure (for example, no list, no dictionary, etc.)

Basically, we want to store a database of the people we are tracking. For each person, we keep the following information:

1. `name` : name of the person (assume this is unique for each person)
2. `home` : place of residence (assume at most one place for each person)
3. `workplace` : place of work (assume at most one place for each person)
4. `caseType` : Normal (`"n"`), Quarantined (`"q"`) or Confirmed (`"c"`)
5. `places` : places visited by the person - it can be zero or many locations.

You are required to implement the following functions:

1. `make_empty_db` : This function takes no arguments and returns an empty database.
2. `add_person` : This function takes a `database` and a person's `name`, `workplace`, `home`, and `caseType` and returns a new database with the entry of the new person inserted.
3. `remove_person` : This function takes in a `database` and a person's `name`. If the person with the given name exists, the function must return a new database with the entry for the specified person removed.
4. `same_home_or_office_as` : This function takes in a `database` and a person's `name`. It should return a tuple of `names` of people (excluding the said person) staying in the same home location or working at the same office location. It can happen that a home location can also be an office location and vice-versa.
5. `add_visited_places` : This function takes in a `database`, a person's `name` and a tuple of places visited by this person to be recorded as `places` in the said person's database entry.
6. `same_visited_places_as` : This function takes in a `database` and a person's `name` to return a tuple of `names` of people (excluding the said person) who have visited the same places as the given person. For simplicity, we do not involve checking `home` and `workplace` as visited places too.
7. `set_case_to_quarantined` : This function takes in a `database` and a person's `name` to set the `caseType` of this person to `"q"` if this person is not already a confirmed case.
8. `set_case_to_confirmed` : This function takes in a database and a person's `name` to set the `caseType` of this person to `"c"`. In addition, this function needs to set the `caseType` of all people who have either visited the same places or work or stay in the same locations as the specified person to `"q"` (Quarantined) if they are not already confirmed cases.

Important: You are advised to read through all the requirements for this question carefully before deciding on the implementation of your database. You are also encouraged to reuse previously defined functions to make your code readable. We will assume the correctness of these functions regardless of how you have implemented them.

Sample execution:

```
>>> db = make_empty_db()

# Alice stays at "H01", works at "W01", and currently has normal status.
>>> db = add_person( db, "Alice", "H01", "W01", "n")

>>> db = add_person( db, "Ben", "H01", "W02", "n")
>>> db = add_person( db, "Cathy", "H03", "W01", "n")
>>> db = add_person( db, "Dennis", "H04", "H03", "n")

# Ben and Alice have the same home location (H01)
# Cathy and Alice have the same workplace (W01)
>>> print( same_home_or_office_as( db, "Alice" ) )
('Ben', 'Cathy')

# Dennis's workplace and Cathy's home are in the same location (H03)
>>> print( same_home_or_office_as( db, "Dennis" ) )
('Cathy',)

# Removing Cathy from the database
>>> db = remove_person( db, "Cathy" )
>>> print( same_home_or_office_as( db, "Alice") )
('Ben',)
>>> print( same_home_or_office_as(db, "Dennis") )
()

>>> db = add_visited_places( db, "Dennis", ("VivoCity", "SAFRA", "Jurong East") )
>>> db = add_visited_places( db, "Ben", ("SAFRA", "NUS") )
# both Ben and Dennis have visited SAFRA
>>> print( same_visited_places_as(db, "Dennis") )
('Ben',)

# Adding John to the database and setting him to Quarantine
>>> db = add_person(db, "John", "H04", "W01", "n" )
>>> print( same_home_or_office_as(db, "John") )
('Alice', 'Dennis')
>>> print( same_visited_places_as(db, "John") )
()
>>> db = set_case_to_quarantined( db, "John" )
Done quarantine: John

# Confirming Dennis
>>> print( same_home_or_office_as(db, "Dennis") )
('John',)
>>> print( same_visited_places_as(db, "Dennis") )
('Ben',)
>>> db = set_case_to_confirm( db, "Dennis")
Done confirm: Dennis
Already quarantined before: John
Done quarantine: Ben
```

Important: Please plan your codes outside the answer box and then copy neatly (with proper indentation) your answer into the answer box.

A. Write the function `make_empty_db`. [1 marks]

```
def make_empty_db():

    return ()
```

B. Explain how you are going to store people in your database. Then, write the function `add_person`. [1 marks]

```
def add_person(db, name, home, workplace, caseType):
    pRec = (name, home, workplace, caseType)
    return db + (pRec,)
```

We are going to store each person as a tuple containing, in the order, name, home, workplace, caseType, and places visited. For places visited, we can either use a tuple to keep all the places visited, or we can simply append each place visited at the end of the person's tuple.

```
# for later uses. You will not be penalized if you do not have
# similar functions
def get_name(pRec):
    return pRec[0]

def get_home(pRec):
    return pRec[1]

def get_workplace(pRec):
    return pRec[2]

def get_case_type(pRec):
    return pRec[3]
```

C. Base on your way to store each person in the database, write the function `remove_person`. [1 marks]

```
def remove_person(db, name):
    temp = db
    db = ()
    for p in temp:
        if name==get_name(p):
            continue
        db = db + (p,)
    return db
```

D. Write the function `same_home_or_office_as`. Note that it can happen that a home location can also be an office location and vice-versa. [1 marks]

```
def find_record(db, name):
    for p in db:
        if name==get_name(p):
            return p

def same_home_or_office_as(db, name):
    pRec = find_record(db, name)
    pHome = get_home(pRec)
    pOffice = get_workplace(pRec)
    result = ()
    for p in db:
        if name == get_name(p):
            continue
        if get_home(p)==pHome or get_workplace(p)==pOffice or \
           get_home(p)==pOffice or get_workplace(p)==pHome:
            result += (get_name(p),)
    return result
```

E. Write the function `add_visited_places`. This should be in accordance to what you declare in Part (B) to implement your database. [1 marks]

```
def add_record(db, pRec):
    return db + (pRec, )

def add_visited_places(db, name, places):
    pRec = find_record(db, name)
    for p in places:
        pRec += (p,)
    db = remove_person(db, name)
    return add_record(db, pRec)
```

F. Write the function `same_visited_places_as`. For simplicity, we do not involved checking `home` and `workplace` as visited places too. [1 marks]

```
def get_visited_places( pRec ):
    return pRec[4:]

def same_visited_places_as(db, name):
    pRec = find_record( db, name )
    places = get_visited_places( pRec )
    result=()
    for person in db:
        for l in get_visited_places(person):
                if l in places and get_name(person) != name:
                        result += (person[0], )
    return result
```

G. Write the function `set_case_to_quarantined`. A case that is already a `"c"` status should not be quarantined, and a case that is already a `"q"` status should not be set again. In these cases, please output messages as shown in the sample execution. [1 marks]

```
def set_case_type(pRec, t):
    rec = pRec[:3] + (t,) + pRec[4:]
    return rec

def set_case_to_quarantined(db, name):
    pRec = (find_record(db, name))
    if get_case_type(pRec) == "c":
        print("Already confirmed before:", name, "-- no quarantined needed")
        return db
    if get_case_type(pRec) == "q" :
        print("Already quarantined before:", name)
        return db
    pRec = set_case_type(pRec, "q")
    db = remove_person(db, name)
    db = add_record(db, pRec)
    print("Done quarantine:", name)
    return db
```

H. Write the function `set_case_to_confirm`. Besides setting a person to a confirm status, this function needs to set all those other people in the database with the same home or office location or visited the same places as the input person to `"q"` if the status of each such a person is not yet `"c"` nor `"q"`. [1 marks]

```
def set_case_to_confirm(db, name):
    pRec = (find_record(db, name))
    pRec = set_case_type(pRec, "c")
    db = remove_person(db, name)
    db = add_record(db, pRec)
    print("Done confirm:", name)
    people_to_quarantine = same_home_or_office_as(db, name) + \
           same_visited_places_as(db, name)
    for p in people_to_quarantine:
        db = set_case_to_quarantined(db, p)
    return db
```

Scratch Paper

— E N D O F P A P E R —