SMU
SINGAPORE MANAGEMENT UNIVERSITY

AY 2020-21 Term 1 Examination

IS111 Intro to programming

INSTRUCTIONS

Please read this page only. Do not open the paper until instructed.

1. This is a **closed-book** examination. Please remove everything from your workspace except for pens, pencils, erasers, wipe outs and stapler before you begin. No use of calculators is allowed.

2. The time allowed for this examination paper is **TWO** hours.

3. This paper comprises **36** pages including the cover page and **FOUR** blank pages (at the end).

4. This examination consists of 2 sections.
   a. **Section A** consists of 10 multiple choice questions (MCQ). Each MCQ is worth 2 marks. For each question, select the **BEST** choice and put a ✓ inside the box. For example:
      ☑ A. Coding is AWESOME!
   b. **Section B** consists of 7 short answer questions.

5. You are required to write all answers (both sections **A** and **B**) on this examination paper.

6. You can use the blank pages to write your answers if you need extra space. Label your question numbers accordingly.

7. You must **return** all parts of this examination paper to the invigilators. Missing examination paper or parts thereof will be considered cheating.

	Marks	Awarded
Section A (MCQ):	20	
Question 1	9	
Question 2	9	
Question 3	8	
Question 4	7	
Question 5	7	
Question 6	5	
Question 7	5	
TOTAL	70	

Section A

1. **[Difficulty: *]** Which one of the following is a valid variable name (i.e., the name will not cause an error)?

   ☐ A. `True`

   ☐ B. `A-Name`

   ☐ C. `7_dwarfs`

   ☐ D. `$money`

   ☐ E. `GST_RATE`

2. **[Difficulty: *]** What is the output when you run the following program?

```
1  count = 10
2  for i in range(0, 6):
3      count = count - 1
4  print(count)
```

   ☐ A. `0`

   ☐ B. `4`

   ☐ C. `5`

   ☐ D. `6`

   ☐ E. `10`

3. **[Difficulty: *]** What is the output when you run the following program?

```
1  my_str = '0123456789'
2  if 0 in [ my_str ]:
3      print(0, end='')
4
5  if '1' in my_str:
6      print('1', end='')
7
8  if '24' in my_str:
9      print('24', end='')
```

☐ A. 0

☐ B. 1

☐ C. 01

☐ D. 0124

☐ E. 124

4. **[Difficulty: *]** What is the output when you run the following program?

```
1  my_list = [10, 9, 8, 7, 6]
2
3  print(my_list[len(my_list) - 1])
4  print(my_list[0:2])
5  print(my_list[:1])
```

☐ A.
```
6
[10, 9]
[10]
```

☐ B.
```
[6]
[10, 9]
[10]
```

☐ C.
```
[6]
[10, 9]
10
```

☐ D.
```
6
10, 9
10
```

☐ E.
```
6
[10, 9]
10
```

5. **[Difficulty: *]** Given the following code:

```
1  def test(number, string):
2      number = string
3      result = number + string
4
5  n = 1
6  s = 'X'
7  answer = test((n, s))
8  print(n, s, answer)
```

Which statement below is correct?

☐ A. Line 2 has an error because we cannot assign a string to a number.

☐ B. Line 3 has an error because we cannot add a number and a string.

☐ C. Line 7 has an error because the function `test` needs 2 parameters, but only one is sent in.

☐ D. Line 7 has an error because the function `test` does not return any value.

☐ E. There is no error in the code.

6. **[Difficulty: **]** What is the output when you run the following program?

```
1  num = 9
2
3  if str(num) == '9':
4      print("Gryffindor")
5
6  if num == int('3' + '6'):
7      print("Hufflepuff")
8
9  if str(num / 3) == str(3):
10     print("Ravenclaw")
11 else:
12     print("Slytherin")
```

☐ A.
```
Gryffindor
Hufflepuff
Ravenclaw
```

☐ B.
```
Gryffindor
Ravenclaw
```

☐ C.
```
Hufflepuff
Ravenclaw
```

☐ D.
```
Gryffindor
Hufflepuff
Slytherin
```

☐ E.
```
Gryffindor
Slytherin
```

7. **[Difficulty: **]** Given the following code:

```
1   def modify_1(my_list):
2       my_list = []
3
4   def modify_2(my_list):
5       my_list[0] = 'Zeus'
6
7   def modify_3(my_list):
8       your_list = my_list
9       your_list.append(['Hades'])
10
11  def modify_4(my_list):
12      my_value = my_list[0]
13      my_value = my_value + '2'
14
15  a_list = ['Apollo']
16  modify_1(a_list)
17  print(a_list)
18
19  a_list = ['Apollo']
20  modify_2(a_list)
21  print(a_list)
22
23  a_list = ['Apollo']
24  modify_3(a_list)
25  print(a_list)
26
27  a_list = ['Apollo']
28  modify_4(a_list)
29  print(a_list)
```

What will be the output?

☐ A.
```
[]
['Zeus']
['Apollo']
['Apollo']
```

☐ B.
```
[]
['Zeus', 'Apollo']
['Apollo', 'Hades']
['Apollo2']
```

☐ C.
```
['Apollo']
['Zeus']
['Apollo', 'Hades']
['Apollo']
```

☐ D.
```
[]
['Zeus']
['Apollo', ['Hades']]
['Apollo2']
```

☐ E.
```
['Apollo']
['Zeus']
['Apollo', ['Hades']]
['Apollo']
```

8. **[Difficulty: **]** What is the output when you run the following program?

```
01  def do_magic(x):
02      return x[0] + x[1]
03
04  list_1 = [1, 3, 5]
05  print(do_magic(list_1))
06
07  list_2 = [[1], [3], [5]]
08  print(do_magic(list_2))
09
10  my_str = '135'
11  print(do_magic(my_str))
```

☐ A.
```
4
[1, 3]
13
```

☐ B.
```
[1, 3]
[[1], [3]]
13
```

☐ C.
```
[1, 3]
[[1], [3]]
4
```

☐ D.
```
4
[[1], [3]]
13
```

☐ E. There is an execution error.

9. **[Difficulty: ***]** Given the following code:

```
1   def is_valid(my_str):
2       digits = '0123456789'
3       letters = 'abcdefghijklmnopqrstuvwxyz' # A string containing all 26 letters.
4       count_1 = 0
5       for d in digits:
6           count_2 = 0
7           for ch in my_str:
8               if ch == d:
9                   count_2 += 1
10          if count_2 >= 2:
11              count_1 += 1
12
13      if count_1 < 5:
14          return False
15
16      for l in letters:
17          for ch in my_str:
18              if ch == l:
19                  return True
20
21      return False
```

Which of the following `print()` statement will print `True`?

☐ A. `print(is_valid('01234567890abcdefghijklmnopqrstuvwxyz'))`

☐ B. `print(is_valid('012345xyz'))`

☐ C. `print(is_valid('a-12345-54321'))`

☐ D. `print(is_valid('00,11,22,33,44'))`

☐ E. None of the above.

10. **[Difficulty: ***]** A function named `check_numbers()` takes in a list of integers as its only parameter. The function returns `True` if each number in the list is either a divisor or a multiple of another number in the list. Otherwise it returns `False`. If the list is empty, the function returns `True`.

    You can assume that the given list does not contain any duplicate numbers. You can also assume that none of the integers in the list is 0.

    For example, here are some calls to the function and their expected results:

    - `check_numbers([4, 8, 12])` returns `True`
        - 4 is a divisor of 8 and 12; 8 is a multiple of 4; 12 is a multiple of 4.
    - `check_numbers([3, 1, 2])` returns `True`
        - 3 is a multiple of 1; 1 is a divisor of 3 and 2; 2 is a multiple of 1.
    - `check_numbers([4, 2, 6, 7])` returns `False`
        - 7 is not a divisor or a multiple of any other number in the list.
    - `check_numbers([2])` returns `False`
        - Because 2 is the only number in the list, it is not a divisor or a multiple of any other number in the list.
    - `check_numbers([])` returns `True`
        - If the list is empty, the function always returns `True`.

    Which of the implementation(s) below of `check_numbers()` is(are) CORRECT?

I.
```
def check_numbers(my_list):
    for n in my_list:
        for m in my_list:
            if n != m:
                if n % m != 0 or m % n != 0:
                    return False
    return True
```

II.
```
def check_numbers(num_list):
    count = 0
    found = False
    for n in num_list:
        for m in num_list:
            if n != m and not found:
                if n % m == 0 or m % n == 0:
                    found = True
                    count += 1

    if count == len(num_list):
        return True
    else:
        return False
```

III.
```
def check_numbers(num_list):
    for i in range(len(num_list)):
        is_valid = False
        for j in range(len(num_list)):
            if j != i:
                if num_list[i] % num_list[j] == 0 or num_list[j] % num_list[i] == 0:
                    is_valid = True

        if not is_valid:
            return False

    return True
```

☐	A.	I only
☐	B.	II only
☐	C.	III only
☐	D.	I, II and III are all correct
☐	E.	None of I, II or III is correct

Section B

Question 1 [9 marks, Difficulty: *]

In `q1.py`, write a function called `print_digits()` that takes in a parameter called `text` of type `str`.

The function does the following:

a. For every digit character (`'0'` – `'9'`) in `text`, it prints out the digit character as it is.
b. For every non-digit character (e.g., `'a'` or `'$'`) in `text`, it prints out a hex/hash character (i.e.,`'#'`).

For example, given the following script (`q1_test.py`):

```
import q1

print('--')
q1.print_digits('12a4eu$')
q1.print_digits('')
q1.print_digits('#90=')
print('--')
```

Running the script gives the following output:

```
C:\exam>python q1_test.py
--
12#4###

#90#
--
```

Write your answer in the box below.

```
# q1.py
```

Question 2 [9 marks, Difficulty: *]

In q2.py, write a function called count_decimal_places() that takes in a parameter called num_value of type str. num_value is a string that contains a valid whole number (e.g., num_value is "3" or "-10") or a valid decimal number (e.g., num_value is "-5.14" or "9.64").

The function returns the number of decimal places in num_value (i.e., the number of digits to the right of the decimal point). In the case when num_value contains a whole number, the function returns 0.

For example, given the following script (q2_test.py):

```
import q2

print(q2.count_decimal_places('100'))
print(q2.count_decimal_places('11.1234'))
print(q2.count_decimal_places('3.0'))
print(q2.count_decimal_places('3.12345678908'))
print(q2.count_decimal_places('3.'))
```

Running the script gives the following output:

```
C:\exam>python q2_test.py
0
4
1
11
0
```

Write your answer in the box below.

```
# q2.py
```

Question 3 [8 marks]

Part A [Difficulty: *]

Implement a function called `fraction_compare()`. The function takes in two parameters:

- `fraction_1` (type: `tuple`) : A tuple in the form of `(a, b)` where `a` is the numerator and `b` is the denominator.
- `fraction_2` (type: `tuple`) : A tuple in the form of `(c, d)` where `c` is the numerator and `d` is the denominator.

For example,

- the fraction $\frac{3}{4}$ can be represented by `(-3, -4)`, `(6, 8)`, `(-6, -8)`, etc.
- the fraction $-\frac{3}{4}$ can be represented by `(-3, 4)`, `(-6, 8)`, `(6, -8)`, etc.

You can assume that `a`, `b`, `c` and `d` are all integers, and `b` and `d` are not 0.

The function returns an `int` value of `1`, `-1` or `0` based on the result value below:

$$\text{result} = \frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd}$$

- If result > 0, then the function returns `1` (type: `int`).
- If result < 0, then the function returns `-1` (type: `int`).
- If result is zero, then the function returns `0` (type: `int`).

Note:

- Your code is **NOT** allowed to create any float values. E.g., if you compute a/b, it is a float, and this is not allowed. Similarly, if you compute (a * d – b * c) / (b * d), it is also a float and is not allowed.
- You are **NOT** allowed to use the `fractions` library.

For example, given the following script (`q3a_test.py`):

```
import q3

print(q3.fraction_compare((3, 4), (1, 2))) # 3/4 > 1/2
print(q3.fraction_compare((1, 4), (6, 7))) # 1/4 < 6/7
print(q3.fraction_compare((1, 6), (1, 6))) # same value
print(q3.fraction_compare((-3, 4), (-1, 2))) # -3/4 < -1/2
print(q3.fraction_compare((1, -2), (-9, 4))) # -1/2 > -9/4
```

Running the script gives the following output:

```
C:\exam>python q3a_test.py
1
-1
0
-1
1
```

Write your answer in the box below:

```
# q3.py
```

Part B [Difficulty: **]

Implement a function called `get_largest_fraction()`. The function takes in a single parameter:

- `fractions` (type: `list`) : It contains a list of tuples. Each tuple is in the form of `(a, b)` where
  - `a` is the numerator, and
  - `b` is the denominator.

  You can assume that both `a` and `b` are integers and `b` is not 0.

The function returns the largest fraction in the `fractions` list. If the list is empty, the function returns an empty list.

Note: You **MUST** use the function that you have defined in Part (A) to help you solve this problem.

For example, given the following script (`q3b_test.py`):

```
import q3

print(q3.get_largest_fraction([(1, 4), (1, 2), (7, 8), (3, 4)]))
print(q3.get_largest_fraction([(1, 4)]))
print(q3.get_largest_fraction([]))
```

Running the script gives the following output:

```
C:\exam>python q3b_test.py
(7, 8)
(1, 4)
[]
```

Write your answer in the box below:

```
# q3.py
# Assume fraction_compare is implemented and in the same file.
```

Question 4 [7 marks, Difficulty: **]

Part (A)
Define a function called `is_compatible()`. The function takes in the following parameters:

- `donor` (type: `str`): The blood type of the person donating blood.). You can assume a valid blood type (e.g. `'A+'`) is used.
- `recipient` (type: `str`): The blood type of the person receiving the blood donation. You can assume a valid blood type is used.

The function returns `True` if the donor's blood type is compatible with the recipient's. Otherwise, it returns `False`. The compatibility table is listed below:

		Donor's blood type							
		O-	O+	B-	B+	A-	A+	AB-	AB+
Recipient's blood type	AB+	✓	✓	✓	✓	✓	✓	✓	✓
	AB-	✓		✓		✓		✓	
	A+	✓	✓			✓	✓		
	A-	✓				✓			
	B+	✓	✓	✓	✓				
	B-	✓		✓					
	O+	✓	✓						
	O-	✓							

For example,

- `is_compatible('A+', 'AB+')` should return `True`
- `is_compatible('A+', 'A+')` should return `True`
- `is_compatible('A+', 'AB-')` should return `False`

Note: To be awarded any marks, you

1. **MUST** use the dictionary data type
2. **MUST NOT** use any form of conditionals (e.g. if-else, if-elif, if-elif-else)
3. **MUST NOT** use list comprehension (e.g., `[(a – 1) for a in num_list]`)

```
# q4.py
```

Part (B)

Define a function called `get_donors()`. The function takes in two parameters:

a. `willing_donors` (type: `list`): A list of tuples. Each tuple represents a person who is willing to donate blood. The tuple is in the form of `(donor_name, donor_blood_type)`. You can assume a valid blood type (e.g. `'A+'`) is used.
b. `recipient` (type: `str`): The blood type of the person receiving the blood donation. You can assume a valid blood type is used.

The function returns a list consisting of willing donors whose blood types are compatible with the recipient's blood type.

Note: You **MUST** use the function that you have defined in Part (A) to help you solve this problem.

For example, given the following script (`q4b_test.py`):

```
import q4

donor_list = [('apple','A+'), ('orange', 'O+'), ('pear', 'AB-')]
print(q4.get_donors(donor_list, 'A+'))
print('----')

donor_list = [('apple','A+'), ('orange', 'A-'), ('pear', 'AB-')]
print(q4.get_donors(donor_list, 'O+'))
print('----')

print(q4.get_donors([], 'A+'))
```

Running the script gives the following output:

```
C:\exam>python q4b_test.py
[('apple', 'A+'), ('orange', 'O+')]
----
[]
----
[]
```

Write your answer in the box below:

```
# q4.py
# Assume is_compatible is implemented and in the same file.
```

Question 5 [7 marks, Difficulty: **]

Examine the following program and list the output when it runs.

- Write 'error' if you think the code will crash (i.e., raise an error) during execution. You should still list the output up to the point of the execution error.
- Write 'nothing' if you think the code will not produce any output.

```
def do_something(my_list):
    print(my_list[len(my_list) // 2])
    my_list[len(my_list) - 1] = 'golden'
    if len(my_list) > 4:
        return True
    return False

my_list = ['blue', 'white', 'red']
index = 1
while my_list[index] != 'golden':
    if index < 3:
        print(my_list[0])
        my_list = my_list[index:]
        my_list.append('cyan')
        my_list.append('pink')
    elif do_something(my_list):
        print(my_list[1])
    else:
        my_list = my_list + ['yellow', 'orange']
        print(my_list[index + 1])
    index += 1
print(index)
```

Output when the program is executed (what you see on the console window only)

Question 6 [5 marks, Difficulty: ***]

Implement a function called `get_social_people(meetups, m, n)`. This function takes in the following parameter:

- `meetups` (type: `list`): This is a list of tuples, where each tuple contains `(person1, person2, date)`, representing a meetup between `person1` and `person2` on a given `date`. Here, `person1` and `person2` are both strings. `date` is a **negative integer** representing the date of the meetup relative to the current date. E.g., if `date` is -3 and today is Nov 20, it means `person1` and `person2` met up on Nov 17.
- `m`: This is a positive integer.
- `n`: This is another positive integer.

The function returns the list of people who have met at least `m` **different** people in the past `n` days relative to the current date (i.e., the meetup date is `-1`, `-2`, …, or `-n`).

Note the following assumptions about the tuples in the list `meetups`:

- Each tuple contains two different names. I.e., you will not have anything like `('eric', 'eric', -3)`.
- The order of the two people in a tuple does not matter. E.g., `('joe', 'eric', -1)` is the same as `('eric', 'joe', -1)`.
- The same pair of people may have met more than once on different days and therefore appear more than once in the list. E.g., you might see both `('jack', 'eric', -4)` and `('eric', 'jack', -3)` appearing in the list.
- The same pair of people may also have met more than once on the same day and therefore appear more than once in the list. E.g., you might see `('jack', 'eric', -4)` appearing twice in the list, or both `('jack', 'eric', -4)` and `('eric', 'jack', -4)` appearing in the list.
- The tuples in the list are **NOT** sorted in any chronological order.

For example, if the list is `[('joe','eric',-1), ('joe', 'eric', -2), ('tim','eric',-2), ('eric','jack',-3), ('jack','george',-7), ('jack','cindy',-2), ('jack','eric',-4)]`, this means:

1. Joe and Eric met up both 1 day ago and 2 days ago.
2. Tim and Eric met up 2 days ago.
3. Eric and Jack met up 3 days ago.
4. Jack and George met up 7 days ago.
5. Jack and Cindy met up 2 days ago.
6. Jack and Eric also met up 4 days ago.

Example 1: If the function is invoked like this:

```
get_social_people([('joe','eric',-1), ('joe', 'eric', -2), ('tim','eric',-2),
('eric','jack',-3), ('jack','george',-7), ('jack','cindy',-2), ('jack','eric',-4)], 2,
3)
```

The function returns either `['eric', 'jack']` or `['jack', 'eric']`. This is because only Eric and Jack each met up with at least 2 different friends in the past 3 days. (Eric met up with Joe, Tim and Jack in the past 3 days, while Jack met up with Eric and Cindy in the past 3 days.) Note that although Joe met up with Eric *twice* in the past 3 days, it's the same friend that Joe met up with, and therefore it's counted as Joe only meeting up with 1 friend.

Example 2: If the function is invoked like this:

```
get_social_people([('joe','eric',-1), ('joe', 'eric', -2), ('tim','eric',-2),
('eric','joe',-1), ('eric','jack',-3), ('jack','george',-7), ('jack','cindy',-2),
('jack','eric',-4)], 3, 5)
```

The function returns `['eric']` because only Eric met up with at least 3 different friends (i.e., Joe, Tim and Jack) in the past 5 days.

Example 3: If the function is invoked like this:

```
get_social_people([('joe','eric',-1), ('joe', 'eric', -2), ('tim','eric',-2),
('eric','joe',-1), ('eric','jack',-3), ('jack','george',-7), ('jack','cindy',-2),
('jack','eric',-4)], 3, 1)
```

The function returns [ ] because nobody met up with at least 3 different friends in the past 1 day.

Write your answer in the box below:

```
# q6.py
```

Question 7 [5 marks, Difficulty: ***]

q7.py (on Page 29) is a buggy implementation of the last question from Lab Test 1. Identify and correct **ALL** execution and logic errors (i.e., errors that cause the program to behave incorrectly when executed). An error has been identified for you on page 29.

Note: 1 mark will be deducted for every incorrect error that you have identified. The minimum score for this question is 0 mark.

The description of the question is as follows:

Implement a function `trace_contacts_2()` that works as follows:

- The function takes in 4 parameters:
  - `patient` (type: `str`): the name of a person who just starts to develop symptoms on the current day.
  - `history` (type: `list`): a list of tuples that stores the meeting history of people in the community. Specifically, each tuple contains three elements:
    - `p1` (type: `str`): a person's name
    - `p2` (type: `str`): another person's name (which is always different from `p1`)
    - `day` (type: `int`): a *negative* integer indicating the day of the meeting relative to the current day (i.e., the day when `patient` develops symptoms)

    For example, the tuple `("Jason", "Gideon", -3)` means Jason met Gideon three days ago. (I.e., if the current date is September 23, 2030, which is the date when `patient` develops symptoms, then Jason and Gideon met on September 20, 2030.)
  - `m` (type: `int`): a positive integer indicating how many days a person is infectious but asymptomatic.
  - `n` (type: `int`): a positive integer indicating how many days it takes for a person to develop symptoms from the day when he catches the virus. You can assume that `n>m`.

catches the virus	non-infectious period			infectious period				develops symptoms and stops meeting people
$-n$ (n days before Day 0)	$-(n-1)$	...	$-(m+1)$	$-m$ (m days before Day 0)	$-(m-1)$	...	-1	Day 0

- The function returns a list of strings, which are the names of those people who have caught the virus either directly or indirectly from `patient`. This returned list should not contain any duplicate elements.

Note:

- The same pair of people could meet each other on different dates, e.g., `('A', 'B', -3)` and `('A', 'B', -2)` may both appear in `history`.
- All dates in `history` (i.e., the negative integers) are with respect to the original `patient`'s Day 0.
- The original `patient` could appear anywhere in the meeting history (not necessarily in the first tuple).
- The tuples in `history` are not necessarily chronologically ordered.

For example,

- Suppose the variable `history` stores the following list:

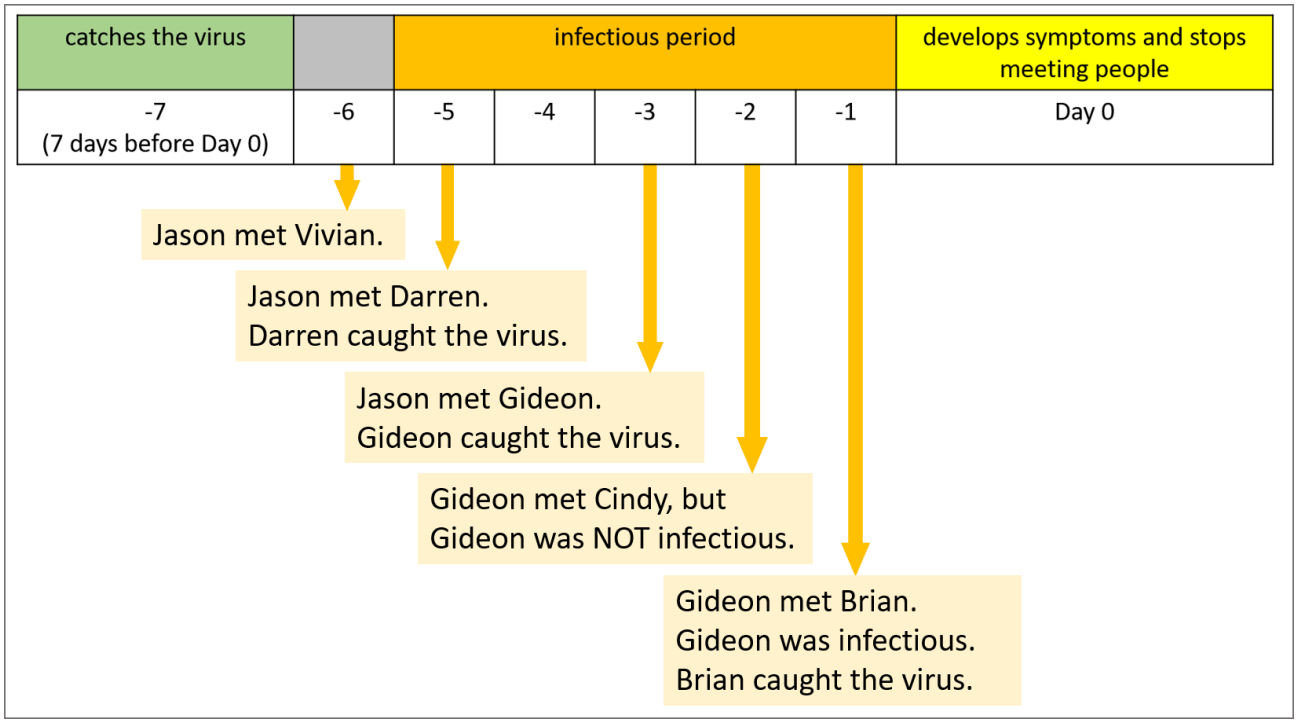
```
history = [("Jason", "Gideon", -3),
           ("Zac", "Yacob", -3),
           ("Gideon", "Brian", -1),
           ("Cindy", "Gideon", -2),
           ("Darren", "Jason", -5),
           ("Jason", "Vivian", -6)]
```

  Suppose `m` is `5` and `n` is `7`.

  Then `trace_contacts_2("Jason", history, 5, 7)` should return

  `["Gideon", "Brian", "Darren"]` (or a list with these three elements in any order)

  This is because Gideon and Darren both met Jason during Jason's infectious period (which is between 5 days ago and 1 day ago), and Brian met Gideon during Gideon's infectious period (which started 1 day ago), given that Gideon has already caught the virus. This is illustrated in the following diagram:



- Suppose the variable history stores the following list:

```
history = [("B", "A", -5),
           ("C", "A", -1),
           ("B", "C", -2)]
```

  Then `trace_contacts_2("A", history, 5, 7)` should return `["B", "C"]` or `["C", "B"]`. Note that here C is listed only once in the returned list, although C met both B during B's infectious period and A during A's infectious period.

```
# q7.py
# spot your errors from this point onwards
def generate_timeline(history, n)  # mistake 1
define generate_timeline(history, n):
    '''
    This function generates a timeline of all the meetups.

    Parameters:
        history (type: list):
                A list of tuples that stores the meeting history.
                Each tuple contains three elements:
                1. p1 (type: str): a person's name
                2. p2 (type: str): another person's name (always different from p1)
                3. day (type: int): a negative integer indicating the
                        day of the meeting relative to the current day
                        (which is the day when the patient develops symptoms)

        n (type: int): A positive integer indicating how many days it takes for a person
                to develop symptoms from the day when he catches the virus.



    Returns:
        A new list of lists that stores the meeting history in chronological order
        from the day when the patient catches the virus (which is n days before the
        current day) to 1 day before the current day.

        For example, if history contains
            [
                ("Z", "X", -4),
                ("D", "F", -1),
                ("C", "D", -3),
                ("A", "C", -5),
                ("B", "A", -5),
                ("A", "X", -4),
                ("E", "D", -2),
                ("X", "A", -6),
                ("Y", "A", -6)
            ]

        and n is 7, this function returns the following list:

            [   [],                         # day on which patient got infected (n days ago).
                [('X', 'A'), ('Y', 'A')],  # 1 day after patient got infected (n-1 days ago).
                [('A', 'C'), ('B', 'A')],  # 2 day after patient got infected.
                [('Z', 'X'), ('A', 'X')],  # 3 day after patient got infected.
                [('C', 'D')],              # 4 day after patient got infected.
                [('E', 'D')],              # 5 day after patient got infected.
                [('D', 'F')]               # 6 day after patient got infected.
            ]

    '''
```

```
58
59
60      timeline = []    # the list to be returned
61
62
63      one_day = []
64
65
66      for i in range(n):
67
68
69          timeline.append(one_day)
70
71
72      for meetup in history:
73
74
75          p1 = meetup[0]
76          p2 = meetup[1]
77          index = meetup[2]
78
79
80          if index >= -n:
81
82
83              timeline[index] += (p1, p2)
84
85
86      return timeline
87
88
89
90  def get_infected_people(timeline, target, start_day, inactive_interval):
91      '''
92      This function returns the list of people who are directly
93      infected by target (who is an infectious person).
94
95      Parameters:
96          timeline (type: list):
97                  A list of lists that stores the meeting history in chronological order
98                  from the day when the original patient catches the virus to 1 day before
99                  the current date. It is the list returned by the function
100                 generate_timeline().
101
102         target (type: str): The name of a person who has been infected (who may or may not
103                 be the original patient).
104
105         start_day (type: int): The first infectious day of target, relative to the day when
106                 the original patient got infected.
107
108         inactive_interval (type: int): How many days the virus will be dormant before
109                 it becomes infectious.
110
111     Returns:
112         A new list of tuples that stores people directly infected by target, and the
113         first infectious days of these people.
114
115         For example, if timeline contains
116             [   [],                         # day on which patient got infected.
117                 [('X', 'A'), ('Y', 'A')],   # 1 day after patient got infected.
```

```
                [('A', 'C'), ('B', 'A')],   # 2 day after patient got infected.
                [('Z', 'X'), ('A', 'X')],   # 3 day after patient got infected.
                [('C', 'D')],               # 4 day after patient got infected.
                [('E', 'D')],               # 5 day after patient got infected.
                [('D', 'F')]                # 6 day after patient got infected.
            ]

        and if target is 'A',
        start day is 2 (meaning that 'A' becomes infectious 2 days after the original
        patient got infected),
        inactive_interval is 3,
        then this function will return:
             [('C', 5), ('B', 5), ('X', 6)]

        For example, ('C', 5) is in the returned list because 'C' met up with 'A' when
        'A' is infectious, and 'C' will start to be infectious 3 days after 'C' met 'A',
        which is 5 days after the original patient got infected.

    '''

    result = []


    for day in range(start_day, len(timeline)):


        meetups = timeline[day]


        for a_meetup in meetups:


            p1 = a_meetup[0]
            p2 = a_meetup[1]


            if p1 == target:


                result.append((p2, day - inactive_interval))


            elif p2 == target:


                result.append((p1, day - inactive_interval))



    return result
```

```
178
179
180
181 def trace_contacts_2(patient, history, m, n):
182
183
184     timeline = generate_timeline(history, n)
185
186
187     # the number of days where the virus remains dormant
188     inactive_interval = n - m
189
190
191     to_contact_list = [(patient, inactive_interval)]
192
193
194     result = []
195
196
197     for one_contact in to_contact_list:
198
199
200         name = one_contact[0]
201
202
203         start_day = one_contact[1]
204
205
206         if name not in result:
207
208
209             result.append(name)
210
211
212
213             to_contact_list = get_infected_people(timeline, name, start_day,
214                     inactive_interval)
215
216
217
218
219     return result
220
```

END OF PAPER.
ENJOY YOUR HOLIDAY!

THIS PAGE IS INTENTIONALLY LEFT BLANK.

THIS PAGE IS INTENTIONALLY LEFT BLANK.

THIS PAGE IS INTENTIONALLY LEFT BLANK.

THIS PAGE IS INTENTIONALLY LEFT BLANK.