|  |  |
| --- | --- |
| SMU-logo-2005-June | **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.
   1. **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! |

* 1. **Section B** consists of 7 short answer questions.

1. You are required to write all answers (both sections **A** and **B**) on this examination paper.
2. You can use the blank pages to write your answers if you need extra space. Label your question numbers accordingly.
3. 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)?

E

|  |  |
| --- | --- |
| A. | True |
| B. | A-Name |
| C. | 7\_dwarfs |
| D. | $money |
| E. | GST\_RATE |

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

|  |  |
| --- | --- |
| 1  2  3  4 | count = 10  Count = 10 – 6 = 4  I = 0,1,2,3,4,5  for i in range(0, 6): count = count - 1  print(count) |

B

|  |  |
| --- | --- |
| A. | 0 |
| B. | 4 |
| C. | 5 |
| D. | 6 |
| E. | 10 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | my\_str = '0123456789' if 0 in [ my\_str ]:  print(0, end='')  if '1' in my\_str: print('1', end='')  if '24' in my\_str: print('24', end='') |

B

|  |  |
| --- | --- |
| A. | 0 |
| B. | 1 |
| C. | 01 |
| D. | 0124 |
| E. | 124 |

|  |  |
| --- | --- |
| 1  2  3  4  5 | my\_list = [10, 9, 8, 7, 6]  print(my\_list[len(my\_list) - 1]) print(my\_list[0:2])  print(my\_list[:1]) |

A

6

[10, 9]

[10]

|  |  |
| --- | --- |
| 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,’X’)

Number = ‘X’

Result = ‘X’ + ‘X’

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

Which statement below is correct?

C

|  |  |
| --- | --- |
| 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. |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | num = 9  if str(num) == '9': print("Gryffindor")  if num == int('3' + '6'): print("Hufflepuff")  (9/3) = 3.0  ‘3.0’ == ‘3’  if str(num / 3) == str(3): print("Ravenclaw")  else:  print("Slytherin") |

Gryffindnor

Slytherin

|  |  |
| --- | --- |
| A. | Gryffindor  Hufflepuff Ravenclaw |
| B. | Gryffindor Ravenclaw |
| C. | Hufflepuff Ravenclaw |
| D. | Gryffindor Hufflepuff Slytherin |
| E. | Gryffindor Slytherin |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | def modify\_1(my\_list): my\_list = []  [‘Apollo’]  [‘Zeus’]  def modify\_2(my\_list): my\_list[0] = 'Zeus'  def modify\_3(my\_list): your\_list = my\_list your\_list.append(['Hades'])  [‘Apollo’]  [‘Apollo’, [‘Hades’]]  def modify\_4(my\_list): my\_value = my\_list[0] my\_value = my\_value + '2'  My\_value = my\_list[0] = Apollo  a\_list = ['Apollo'] modify\_1(a\_list) print(a\_list)  [‘Apollo’]  Print: [‘Apollo’]  a\_list = ['Apollo'] modify\_2(a\_list) print(a\_list)  [‘Apollo’] -> [‘Zues’]  Print: [‘Zeus’]  [‘Apollo’] -> [‘Apollo’, [‘Hades’]]  Print: [‘Apollo’, [‘Hades’]]  a\_list = ['Apollo'] modify\_3(a\_list) print(a\_list)  [‘Apollo’] -> [‘Apollo’, [‘Hades’]]  Print: [‘Apollo’]  a\_list = ['Apollo'] modify\_4(a\_list)  print(a\_list) |

What will be the output?

E

|  |  |
| --- | --- |
| 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'] |

[[1], [3], [5]] -> [1] + [3] = [1, 3]

[1, 3, 5] -> 4

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | def do\_magic(x): return x[0] + x[1]  ‘135’ -> ‘13’  list\_1 = [1, 3, 5] print(do\_magic(list\_1))  4  list\_2 = [[1], [3], [5]]  [1, 3]  print(do\_magic(list\_2))  my\_str = '135' print(do\_magic(my\_str))  13 |

A

|  |  |
| --- | --- |
| 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. |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | def is\_valid(my\_str): digits = '0123456789'  letters = 'abcdefghijklmnopqrstuvwxyz' # A string containing all 26 letters. count\_1 = 0  My\_str = ‘a-12345-54321’  Count\_1 = 0,1,2,3,4,5 # Increment for each 2 same digits  For digits:  Count\_2 = 0  For ch in my\_str:  If ch == d:  Count 2 += 1  If count\_2 >= 2  Count\_1 += 1  For letters:  Ch in mystr:  If Ch == 1 # True  Return True  for d in digits: count\_2 = 0  for ch in my\_str: if ch == d:  count\_2 += 1 if count\_2 >= 2:  count\_1 += 1  if count\_1 < 5: return False  for l in letters: for ch in my\_str:  if ch == l:  return True  return False |

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

C

False -Line14

|  |  |
| --- | --- |
| A. | print(is\_valid('01234567890abcdefghijklmnopqrstuvwxyz'))  False -Line14 |
| B. | print(is\_valid('012345xyz')) |
| C. | print(is\_valid('a-12345-54321'))  True |
| D. | print(is\_valid('00,11,22,33,44'))  False -Line21 |
| E. | None of the above. |

1. **[ 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?

[] - > True

[2] -> True

|  |  |  |
| --- | --- | --- |
| 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 |  |

[] - > True

[2] -> False

[3,1,2] -> False

|  |  |
| --- | --- |
| 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 |

[] - > True

[2] -> False

[3,1,2] -> True

[4,2,6,7] -> False

C

|  |  |
| --- | --- |
| 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:

1. For every digit character ('0' – '9') in text, it prints out the digit character as it is.
2. 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.**

def print\_digits(text):

final\_text = ‘’

for ch in text:

if ch.isnumeric():

final\_text += ch

else:

final\_text += ‘#’

print(final\_text)

# 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.**

def count\_decimal\_places(num\_value):

try:

return len(num\_value.split(‘.’)[1])

except:

return 0

def count\_decimal\_places(num\_value):

if '.' in num\_value:

return len(num\_value.split(‘.’)[1])

else:

return 0

# 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 3

o

4

can be represented by (-3, -4), (6, 8), (-6, -8), etc.

the fraction − 3

o

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:

* 1. c

result = −

* 1. d

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

def fraction\_compare(fraction\_1,fraction\_2):

a = fraction\_1[0]

b = fraction\_1[1]

c = fraction\_2[0]

d = fraction\_2[1]

result\_numerator = a\*d-b\*c

result\_denominator = b\*d

result = (a\*d-b\*c) // (b\*d)

if result < 0:

return -1

elif result\_numerator == 0:

return 0

else:

return 1

### 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:**

def get\_largest\_fraction(fractions):

max\_fraction = []

if fractions:

max\_fraction = fractions[0]

for fraction in fractions:

If fraction != max\_fraction:

result = fraction\_compare(fraction, max\_fraction)

If result == 1:

max\_fraction = fraction

return max\_fraction

# 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])

def is\_compatible(donor, recipient):

donor\_dict = { ‘O-‘:[‘AB+’,’AB-‘,’A+’,’A-‘,’B+’,’B-‘,’O+’,’O-‘],

‘O+’:[‘AB+’,’A+’,’B+’,’O+’],

‘B-‘:[‘AB+’,’AB-‘,’B+’,’B-‘],

‘B+’:[‘AB+’,’B+’],

‘A-‘:[‘AB+’,’AB-‘,’A+’,’A-‘],

‘A+’:[‘AB+’,’A+’],

‘AB-‘:[‘AB+’,’AB-‘],

‘AB+’:[‘AB+’]

}

return recipient in donor\_dict[donor]

# q4.py

### Part (B)

Define a function called get\_donors(). The function takes in two parameters:

* 1. 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.
  2. 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.

def get\_donors(willing\_donors, recipient):

compatible\_donors = []

for donor in willing\_donors:

if is\_compatible(donor[1], recipient):

compatible\_donors.append(donor)

return compatible\_donors

## 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.

My\_list = [blue,white,red]

Index = 1

While my\_list[index] != ‘golden’:

print(blue)

my\_list = [white,red] + [cyan] + [pink]

Index = 2

NEXT

Print(white)

my\_list = [cyan, pink] + [cyan, pink]

index = 3

NEXT

Index >3: (do\_something)

Len(my\_list) //2 = 2, print(cyan)

My\_list[3], [cyan, pink,cyan, golden]

NEXT

Else:

[cyan, pink,cyan, golden,yellow,orange]

Print(my\_list[4]

Index = 4

NEXT

Index >3: (do\_something)

Len(my\_list) //2 = 3, print(golden)

My\_list[5], [cyan, pink,cyan, golden, yellow, golden]

Return True -> print(my\_list[1])

Index = 5

NEXT

Exit Loop

Print(index)

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)

blue

white

cyan

yellow

golden

pink

5

## 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

def get\_social\_people(meetups,m,n):

person\_meetups = {}

n \*= -1

for meetup in meetups:

person1 = meetup[0]

person2 = meetup[1]

time\_meetup = meetup[2]

if time\_meetup >= n:

if person1 not in person\_meetups:

person\_meetups[person1] = [person2]

elif person2 not in person\_meetups[person1]:

person\_meetups[person1].append(person2)

if person2 not in person\_meetups:

person\_meetups[person2] = [person1]

elif person1 not in person\_meetups[person2]:

person\_meetups[person2].append(person1)

return\_list = []

for main\_person in person\_meetups:

if len(person\_meetups[main\_person]) >= m:

return\_list.append(main\_person)

return return\_list

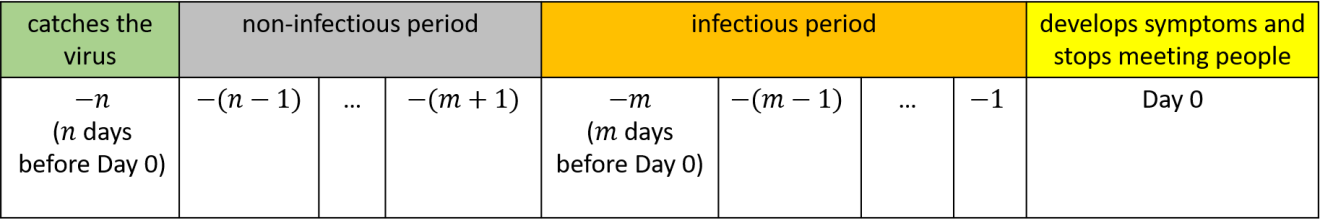
## Question 7 [ 5 marks, Difficulty: \*\*\* ]

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:

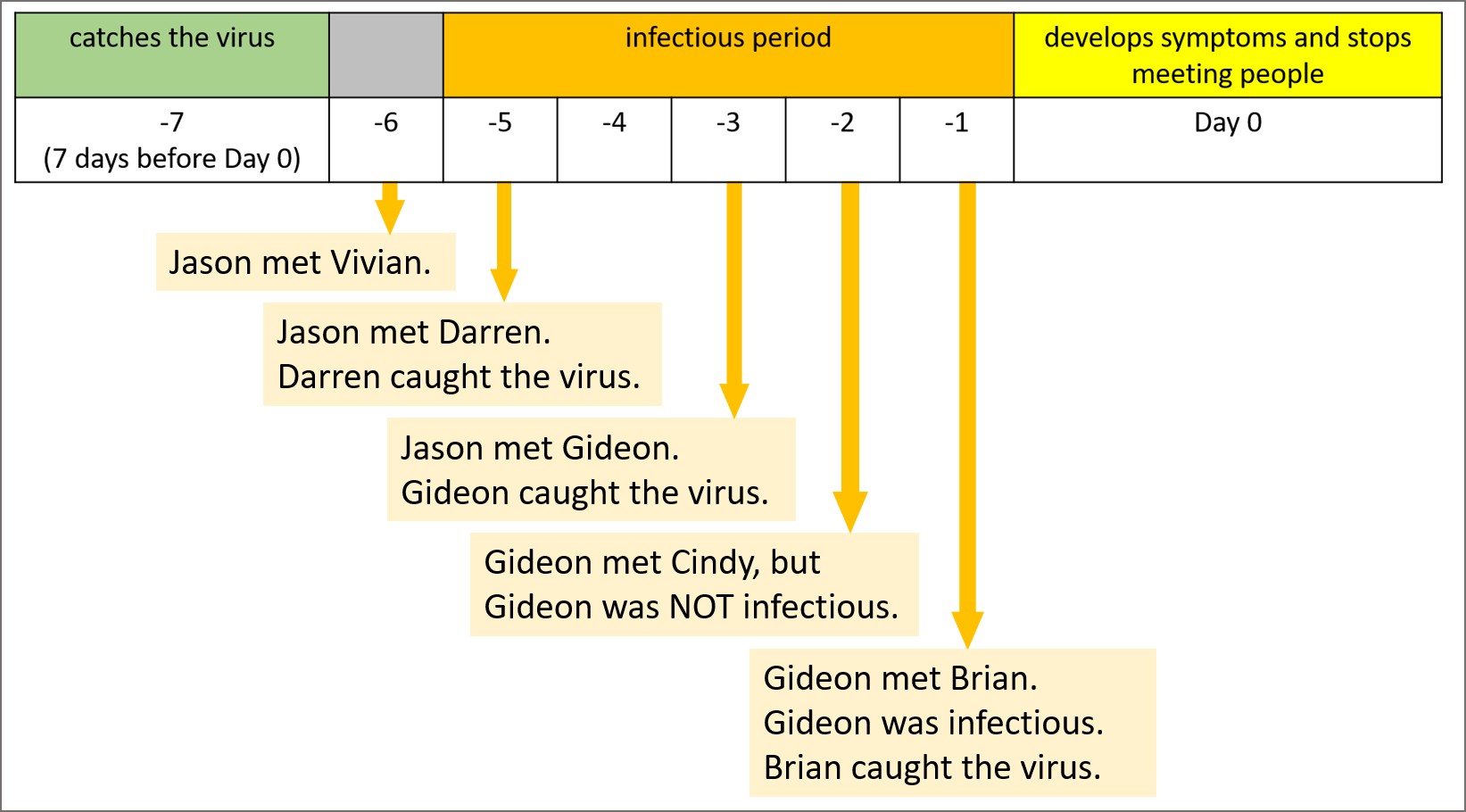
* + - p1 (type: str): a person’s name
    - p2 (type: str): another person’s name (which is always different from p1)
    - day (type: int): a *negative* integer indicating the day of the meeting relative to the current day (i.e., the day when patient develops symptoms)

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.
* 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | # 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  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117 | timeline = [] # the list to be returned one\_day = []  for i in range(n): timeline.append(one\_day)  for meetup in history:  p1 = meetup[0] p2 = meetup[1] index = meetup[2]  if index >= -n:  timeline[index] += (p1, p2) return timeline  def get\_infected\_people(timeline, target, start\_day, inactive\_interval): '''  This function returns the list of people who are directly infected by target (who is an infectious person).  Parameters:  timeline (type: list):  A list of lists that stores the meeting history in chronological order from the day when the original patient catches the virus to 1 day before the current date. It is the list returned by the function generate\_timeline().  target (type: str): The name of a person who has been infected (who may or may not be the original patient).  start\_day (type: int): The first infectious day of target, relative to the day when the original patient got infected.  inactive\_interval (type: int): How many days the virus will be dormant before it becomes infectious.  Returns:  A new list of tuples that stores people directly infected by target, and the first infectious days of these people.  For example, if timeline contains  [ [], # day on which patient got infected.  [('X', 'A'), ('Y', 'A')], # 1 day after patient got infected. |

|  |  |
| --- | --- |
| 118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177 | [('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  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220 | def trace\_contacts\_2(patient, history, m, n): timeline = generate\_timeline(history, n)  # the number of days where the virus remains dormant inactive\_interval = n - m  to\_contact\_list = [(patient, inactive\_interval)] result = []  for one\_contact in to\_contact\_list: name = one\_contact[0]  start\_day = one\_contact[1] if name not in result:  result.append(name)  to\_contact\_list = get\_infected\_people(timeline, name, start\_day, inactive\_interval)  return result |

## END OF PAPER. ENJOY YOUR HOLIDAY!