Student Name: _____ **Email ID:** _____

SMU SINGAPORE MANAGEMENT UNIVERSITY

AY 2019-20 Term 2 Examination

IS111 Introduction to programming

INSTRUCTIONS

1. This examination is open-book. However, you must complete it **independently** without any discussion with others. If you consult anyone on any question, it will be considered cheating and result in the grade 'F'.

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

3. This examination consists of 2 sections.
   a. **Section A** consists of 8 multiple choice questions (MCQ). Each MCQ is worth 3 marks. For each question, select the **BEST** choice as your answer.
   b. **Section B** consists of 5 short answer questions.

4. Download AnswerSheet.txt from eLearn. Type your answers for each question following the instructions in the text file.

5. Before submitting your answers, you must **rename AnswerSheet.txt to your own email ID.** For example, if your SMU email is `tiaoshe.qian.2018@sis.smu.edu.sg`, you should rename the text file to `tiaoshe.qian.2018.txt`

	Marks	Awarded
Section A (MCQ): Questions 1-8	24	
Section B: Question 1	12	
Question 2	4	
Question 3	12	
Question 4	8	
Question 5	10	
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 when the code is run)?

   A. `_I$111_I$_awesome`

   B. `IS111-G1`

   C. `111class`

   D. `my_111_module`

   E. All of the above

2. **[Difficulty: *]** In the following program, which condition must be satisfied for the loop to terminate (i.e., skip Ln 2 and go to Ln 3)?

```
1  for i in range(2, 4, 1):
2      print(i)
3
```

   A. `2 < 4`

   B. `i - 1 > 4`

   C. `2 >= 4`

   D. `i < 2`

   E. `i >= 4`

3. **[Difficulty: **]** If `a` and `b` are both non-empty lists, which of the following code is ALWAYS valid?

   A. `a + b`

   B. `a[0] + b[0]`

   C. `a.extend(b[0])`

   D. None of the above

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

```
1  def do_trick(num1, num2):
2      num2 = num2 + 5
3      return num1
4
5  num1 = 1
6  num2 = 2
7  answer = do_trick(num1, num2)
8  print(answer)
9  print(num2)
```

Which of the following statements is correct?

A. `num1` in Line 1 is the same variable as `num1` in Line 5.

B. The output of `print(answer)` in Line 8 depends on the value of `num2`.

C. `print(num2)` in Line 9 prints `'7'`.

D. A and B only

E. None of the above

5. **[Difficulty: **]** In following code, after Ln 8 finishes executing, which of the following statements is true?

```
1  def do_mystery(word, records):
2      word = 'classmate'
3      records[1] = ('lecturer', 5)
4
5  word = 'student'
6  records = [('student', 50), ('teacher', 5), ('TA', 8)]
7
8  do_mystery(word, records)
9
```

A. The variable `word` has the value `'classmate'`

B. The second element of the list `records` contains a new memory address that points to the tuple `('lecturer', 5)`

C. The function `do_mystery` has created new memory locations for the variables on Ln 5 and Ln 6: the string `word` and the list `records`

D. The function `do_mystery` has only modified the first part of the second element of the list `records` by changing it from `'teacher'` to `'lecturer'`

E. None of the above

6. **[Difficulty: *]** Which of the following statements is true?

```
1  def do_something(x, y):
2      x = y + x
3      return x
4
```

A. `x` and `y` can only be integers

B. `x` and `y` must be the same numerical type

C. The function `do_something` prints the sum of `x` and `y`

D. All of the above

E. None of the above

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

len(str1) <= len(str2)
or 'a' in str1

```
1  str1 = input("Enter str1:")
2  str2 = input("Enter str2:")
3  while len(str1) > len(str2) and 'a' not in str1:
4      str1 = input("Enter str1:")
5      str2 = input("Enter str2:")
6
7  print("Bye!")
```

When the execution of the program reaches Ln 7, which of the following statements must be **TRUE**?

A. `str1` is shorter than `str2` or is of the same length as `str2`

B. `str1` contains `'a'`

C. if `str1` is shorter than `str2`, `str1` must contain `'a'`

D. if `str1` is longer than `str2`, `str1` must contain `'a'`

E. None of the above

8. **[Difficulty: ***]** A function named `compare_numbers()` accepts a string parameter named `two_numbers`, which is consisted of two sets of digits separated by `'#'` (e.g., `'123#542'` or `'98281#023'`).
The function returns `True` if the following conditions are all satisfied:
   - Both numbers consist of only digits.
   - The first number is greater than or equal to the second number.

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

- `compare_numbers('299#280')` should return `True`
- `compare_numbers('02246#2248')` should return `False`
  - The first number is `2246`, which is less than `2248`
- `compare_numbers('22a#00')` should return `False`
  - The first number contains a non-digit character
- `compare_numbers('0103#-99')` should return `False`
  - The second number contains `'-'` which is a non-digit character

Which of the implementation(s) below of `compare_numbers()` is(are) correct?

I.

```
def check_digits(all_digits):
    DIGITS = '0123456789'

    for i in range(len(all_digits)):
        if all_digits[i] not in DIGITS:
            return False
    return True


def compare_numbers(two_numbers):
    index = two_numbers.find('#')
    str1 = two_numbers[:index:]
    str2 = two_numbers[index + 1::]

    if check_digits(str1+str2):
        return int(str1) >= int(str2)
    else:
        return False
```

√

II.

```
def check_digits(all_digits):
    DIGITS = '0123456789'
    for ch in all_digits:
        if ch not in DIGITS:
            return False
        else:
            return True

def compare_numbers(two_numbers):
    index = two_numbers.find('#')
    str1 = two_numbers[:index:]
    str2 = two_numbers[index + 1::]

    if check_digits(str1+str2):
        return int(str1) >= int(str2)
    else:
        return False
```

III.

```
def check_digits(all_digits):
    for i in range(len(all_digits)):
        if 0 <= int(all_digits[i]) <= 9:
            return True
        else:
            return False

def compare_numbers(two_numbers):
    index = two_numbers.find('#')
    str1 = two_numbers[:index:]
    str2 = two_numbers[index+1::]

    if check_digits(str1+str2):
        return int(str1) >= int(str2)
    else:
        return False
```

A. I only

B. II only

C. III only

D. I and III only

E. None of I, II or III is correct

Section B

Question 1 [12 marks; Difficulty: **]

(A) Convert the following `for` loop to a `while` loop

```
word = 'long string'

for ch in word:
    print(ch)
```

```
word = 'long string'
i=0
while i<len(word):
    print(word[i])
    i+=1
```

(B) Convert the following `for` loop to a `while` loop

```
for i in range(100, -1, -2):
    print(i)
```

```
i=100
while i>-1:
    print(i)
    i-=2
```

(C) Write the condition that terminate the following while loop (assuming all variables have been defined). Your answer must not contain the word "not"; in other words, the condition must be fully simplified using DeMorgan's Law.

```
# variable initialization omitted

while count < target or some_found == True and count < 100 or all_found == False:

    # some loop body code omitted
    count += 1
```

while not (C<T or some and count<100 or all = False)

not (C<T or some) and not (count <100 or all=false)

(count >= target and some_found == False) and (count >= 100 and all_found)

Question 2 [4 marks; Difficulty: *]

`q2.py` is a buggy implementation of a program. 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 as an example in AnswerSheet.txt.

The write-up for the program is the following:

Write a program called `q2.py` that prompts the user for three `int` values, *x, y* and *z* (assume correct `int` inputs are given). The program then computes *x* raised to the power of *y*, and then raised to the power of *z,* and displays the result as shown in the sample below. (Note that in the sample, boldfaced ones are user inputs)

A sample run of the program is shown below:

```
C:\exam>python q2.py
Enter x:2
Enter y:3
Enter z:2

Based on the values of 2, 3, and 2, for x, y, and z, respectively, the
result is 64.

C:\exam>
```

```
1  # q2.py
2  x = input('Enter x:')
3  y = input('Enter y:')
4
5  result = x ** y ** z
6
7  print('Based on the values of', x, y, 'and', z, ', for x, y, and z, respectively, the result is', result, '.')
8
9
```

int(int(→ z = int(input('Enter z:'))

print(f'Based on the values of {x}, {y}, and {z}, for x, y, and z, respectivaly, the result is {result}.')

Question 3 [Part A 4 marks, Part B 8 marks; Difficulty: **]

Part (A)
Below is a buggy implementation of the function `extract_substrings()`. Identify and correct **ALL** execution and logic errors (i.e., errors that cause the program to behave incorrectly when executed).

The write-up for the function `extract_substrings()` is the following:

> Define a function called `extract_substrings()`. The function takes in two parameters: `digit_str` (type:`str`) and `substr_len` (type:`int`), where `digit_str` is a string of digits.
>
> The function returns a list of strings that are substrings of `digit_str` of length `substr_len`:
>
> For example,
> - `extract_substrings('228591', 2)` returns `['22', '28', '85', '59', '91']`.
> - `extract_substrings('5048', 3)` returns `['504', '048']`.
> - `extract_substrings('5048', 5)` returns `[]`.

```
1  # A buggy implementation of Part A
2
3  def extract_substrings(digit_str, substr_len):
4      list_to_return = []
5
6      for i in range(len(digit_str) - substr_len + 1):
7          list_to_return.extend(digit_str[i : i + 2])
8
9      return list_to_return
```

Part (B)
Define a function called `add_substring_numbers()`. The function takes in a string of digits, `digit_str`, and returns the sum of all numbers represented by the substrings of `digit_str`. You may assume `digit_str` contains only digits.

You **must** use the function `extract_substrings()` described in Part (A) (assuming implemented correctly) to solve this problem. You do NOT need to define `extract_substrings()`, and can assume it is already correctly defined in the same file.

Below are some examples of the output of `add_substring_numbers()`:
- `add_substring_numbers('5048')` returns `5719` because summing all substring numbers 5 + 0 + 4 + 8 + 50 + 4 + 48 + 504 + 48 + 5048 = 5719. Notice that the whole string is also considered a substring of itself.
- `add_substring_numbers('000')` returns `0`.

```
def add_substring_numbers(digit_str):
    sum = 0
    for i in range(1, len(digit_str)+1):
        substrings = extract_substrings(digit_str, i)
        for s in substrings:
            sum += s

    return sum
```

Question 4 [8 marks; Difficulty: ***]

Implement a function called `get_close_contacts(case_list, contact_network, degree_separation)` for contact tracing during the COVID-19 outbreak. The three parameters are defined as follows:

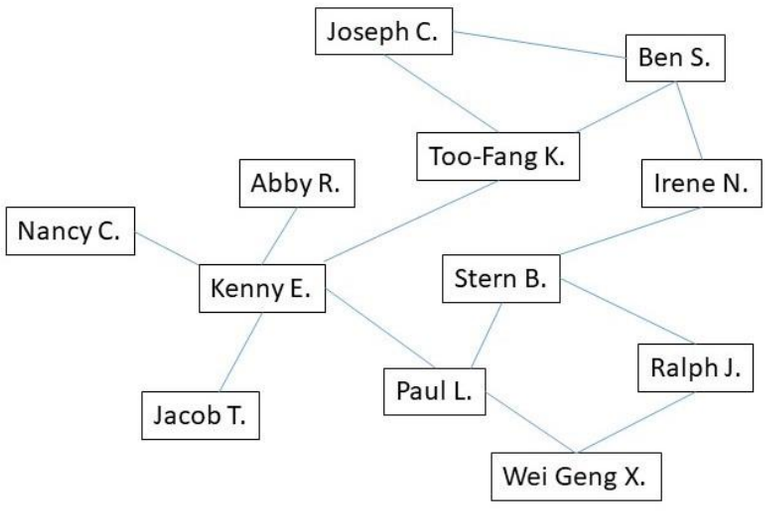
- `case_list` (type: `list`): This list contains the names of the people who tested positive for COVID-19 on a given day
- `contact_network` (type: `dictionary`): This is a dictionary that describes the people who are close contacts in the recent timeframe. The key is the name of a person, and the value of the key is a list containing the names of the close contacts of the person indicated by the key.
- `degree_separation` (type: `int`): This number determines how far down the contact network the function retrieves contact names. If it is 1, only the first-degree contacts are included in the list to return; if it is 2, the first-degree contacts as well as their first-degree contacts (who are second-degree contacts of the original name(s) in `case_list`), if any, are included in the list to return; and so on. You may assume this parameter is greater than `0`.

The function returns the list of the names of all people who are close contacts of all names in `case_list`, based on `contact_network` and `degree_separation`. However, the list returned should not include the original names from `case_list` and should only include unique names. The names in the returned list can be in any order.

An example of `contact_network` is the following:

```
{'Joseph C.': ['Too-Fang K.', 'Ben S.'],
 'Too-Fang K.': ['Joseph C.', 'Ben S.', 'Kenny E.'],
 'Paul L.': ['Kenny E.', 'Stern B.', 'Wei Geng X.'],
 'Nancy C.': ['Kenny E.'],
 'Jacob T.': ['Kenny E.'],
 'Abby R.': ['Kenny E.'],
 'Stern B.': ['Paul L.', 'Irene N.', 'Ralph J.'],
 'Wei Geng X.': ['Paul L.', 'Ralph J.'],
 'Irene N.': ['Stern B.', 'Ben S.'],
 'Ben S.': ['Too-Fang K.', 'Joseph C.', 'Irene N.'],
 'Kenny E.': ['Nancy C.', 'Abby R.', 'Jacob T.', 'Paul L.', 'Too-Fang K.'],
 'Ralph J.': ['Stern B.', 'Wei Geng X.']}
```

Below is the diagram for `contact_network` based on the example above:



If the function is invoked like this, using the example of `contact_network` above:

```
print(get_close_contacts(['Stern B.'], contact_network, 1))
```

The output will be:

```
['Paul L.', 'Irene N.', 'Ralph J.']
```

If the function is invoked like this, using the example of `contact_network` above:

```
print(get_close_contacts(['Stern B.', 'Wei Geng X.'], contact_network, 2))
```

The output will be:

```
['Paul L.', 'Irene N.', 'Ralph J.', 'Ben S.', 'Kenny E.']
```

If the function is invoked like this, using the example of `contact_network` above:

```
print(get_close_contacts(['Irene N.', 'Ben S.'], contact_network, 2))
```

The output will be:

```
['Stern B.', 'Paul L.', 'Ralph J.', 'Joseph C.', 'Too-Fang K.', 'Kenny E.']
```

If the function is invoked like this, using the example of `contact_network` above:

```
print(get_close_contacts(['Sally B.'], contact_network, 4))
```

The output will be:

```
[]
```

Note: `'Sally B.'` does not exist in the network of names.

If the function is invoked like this, using the example of `contact_network` above:

```
print(get_close_contacts(['Kenny E.'], contact_network, 2))
```

The output will be:

```
['Abby R.', 'Nancy C.', 'Jacob T.', 'Paul L.', 'Too-Fang K.', 'Stern B.', 'Wei
Geng X.', 'Ben S.', 'Joseph C.']
```

Please include some comments in your code to help us understand your logic.

```
def get_close_contacts(case_list, contact_network, degree_separation):
    for i in range(degree_separation):
        proxy = []
        for c in case_list:
            proxy += final[c]

        case_list = proxy

    return proxy
```

Question 5 [10 marks; Difficulty: ***]

q5.py is a buggy implementation of Question 5 from Lab Test 2. 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 as an example in AnswerSheet.txt.

The writeup for Question 5 from Lab Test 2 is as follows:

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

- `family_dictionary_str` (type: `str`) : a dictionary of parent-child names in the form of a string
- `parent_name` (type: `str`) : a string of the parent's name to look up in the dictionary

`family_dictionary_str` is a string with the format of "`{'ParentName1':['ChildName1', 'ChildName2', ...], 'ParentName2':['ChildName1', 'ChildName2', ...], ...}`". Below is an example of `family_dictionary_str`:

```
"{'Joe':['Fanny', 'Kate'], 'Pat':['Tommy', 'Joe', 'Will', 'Nick'], 'Owen':[],
'Vicky':['Harry']}"
```

The function returns the list of children names under `parent_name`. If `parent_name` is not found in the keys of the dictionary, the function should return `None`. You may assume valid inputs, such that `family_dictionary_str` if converted to a dictionary has valid value for a dictionary.

Example 1:
If the function is invoked like this with the example of `family_dictionary_str` above:

```
print(lookup_names(family_dictionary_str, 'Joe'))
```

the statement generates the following output:

```
['Fanny', 'Kate']
```

Note: Even though `'Joe'` is also the name of a child of Pat's, the function should return the names of the children under the **parent** with the name `'Joe'`.

Example 2:
If the function is invoked like this with the example of `family_dictionary_str` above:

```
print(lookup_names(family_dictionary_str, 'vicky'))
```

the statement generates the following output:

```
['Harry']
```

Note: Notice that the function is NOT case sensitive. The parent name of `'vicky'` in all lower cases should still look up the names of the children under `'Vicky'`.

Example 3:
If the function is invoked like this with the example of `family_dictionary_str` above:

```
print(lookup_names(family_dictionary_str, 'Will'))
```

the statement generates the following output:

```
None
```

Example 4:
If the function is invoked like this with the example of `family_dictionary_str` above:

```
print(lookup_names(family_dictionary_str, 'Owen'))
```

the statement generates the following output:

```
[]
```

A `q5_test.py` is provided below:

```
import q5

family_dict_str = "{'Joe':['Fanny', 'Kate'], 'Pat':['Tommy', 'Joe', 'Will',
'Nick'], 'Owen':[], 'Vicky':['Harry']} "


result = q5.lookup_names(family_dict_str, 'Owen')
print('Test 1')
print("Expected: []")
print('Actual  :', result)
print()

result = q5.lookup_names(family_dict_str, 'vicky')
print('Test 2')
print("Expected: ['Harry']")
print('Actual  :', result)
print()

result = q5.lookup_names(family_dict_str, 'Will')
print('Test 3')
print("Expected: None")
print('Actual  :', result)
print()

result = q5.lookup_names(family_dict_str, 'Pat')
print('Test 4')
print("Expected: ['Tommy', 'Joe', 'Will', 'Nick']")
print('Actual  :', result)
print()
```

```
# q5.py
# spot your errors from this point onwards
define get_dictionary(family_dictionary_str):

    # this function converts the string of a dictionary into a dictionary.

    # Parameter:
    #   family_dictionary_str (type: str): a string that contains a dictionary

    # Returns:
    #    A dictionary based on the string from the parameter

    # Example:
    #   get_dictionary("{'Annie': ['Ben'], 'Tim': ['Ilsa', 'Gary']}")
    #   will return the dictionary {'Annie': ['Ben'], 'Tim': ['Ilsa', 'Gary']}

# initialize variables
    dict_to_return = {}
    parent_name = ''
    child_name = ''
    # in_str is True when the character ch is inside a name string
    in_str = False
    # in_list is True when the character ch is inside a list
    in_list = False

    # This loops iterates through all characters of the dictionary string one by one
    for ch in family_dictionary_str:

        # The following block sets the flag that indicates
        # the start and end of a name string
        if ch == "'" or in_str == False:
            in_str = True
        elif ch == "'":
            in_str = False

        # The following block builds a parent's or a child's name string
        if in_str == True and not in_list:
            # parent names are stored in all lower case letters in the dictionary
            parent_name += ch.lower()
        elif in_str == False and in_list:
            child_name += ch

        # The following block creates a record in the dictionary
        # with the parent's name and an empty list for the children's name(s)
        elif ch == ':':
            dict_to_return[parent_name] = []

        # The following block sets the flag that indicates
        # the start and end of the child name list
        elif ch == '[':
            in_list = True
        elif ch == ']':
            # at the end of the child name list,
            # the last child name is added to the list
            if not child_name == '':
                dict_to_return[parent_name].extend(child_name)

            # Some variables are reset
            parent_name = ''
            child_name = ''
```

```
        # The following block checks for the end of
        # each child's name string and appends the name to the list
        elif ch == ',' and in_list:
            dict_to_return[parent_name].append(child_name)
            child_name = ''

    return dict_to_return


def lookup_names(family_dictionary_str, parent_name):

    # Convert family_dictionary_str to type dictionary
    family_dict = get_dictionary(family_dictionary_str)

    if parent_name.lower() in family_dict.keys():
        return family_dict[parent_name]
    else:
        return None
```

END OF PAPER.
ENJOY YOUR HOLIDAY AND STAY SAFE!