|  |  |
| --- | --- |
| **Student Name:** | **Email ID:** |
| SMU-logo-2005-June | **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.
   1. **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.
   2. **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,](mailto: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)?

D

|  |  |
| --- | --- |
| A. | \_I$111\_I$\_awesome |
| B. | IS111-G1 |
| C. | 111class |
| D. | my\_111\_module |
| E. | All of the above |

1. **[ 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  2  3 | for i in range(2, 4, 1): print(i)  I = 2, 3 |

E

|  |  |
| --- | --- |
| A. | 2 < 4 |
| B. | i - 1 > 4 |
| C. | 2 >= 4 |
| D. | i < 2 |
| E. | i >= 4 |

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

A

a = [1], b = [2]

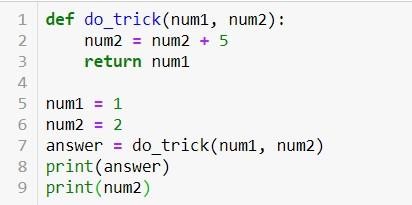
A, B passes

a = [‘a’, 2], b = [1, 2]

A passes

|  |  |
| --- | --- |
| A. | a + b |
| B. | a[0] + b[0] |
| C. | a.extend(b[0]) |
| D. | None of the above |

1. **[ Difficulty: \* ]** Given the following code:



E

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 |

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



D

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

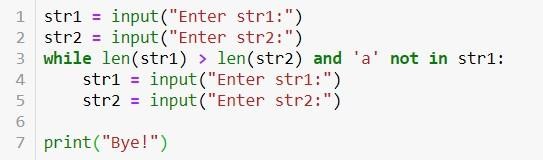
1. **[ Difficulty: \* ]** Which of the following statements is true?

|  |  |
| --- | --- |
| 1  2  3  4 | def do\_something(x, y): x = y + x  return x |

E

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

1. **[ Difficulty: \*\* ]** Given the following code:



!E – All of the above

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 |

1. **[ 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. |  |

|  |  |
| --- | --- |
| II. |  |
| III. |  |

A

Wrong

Wrong

|  |  |
| --- | --- |
| 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: \*\* ]

1. Convert the following for loop to a while loop

word = ‘long string’

iterations = len(word)

i = 0

while i <= iterations:

print(word[i])

I += 1

word = 'long string'

for ch in word: print(ch)

1. Convert the following for loop to a while loop

I = 100

While I > -1:

print(i)

I -= 2

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

1. 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

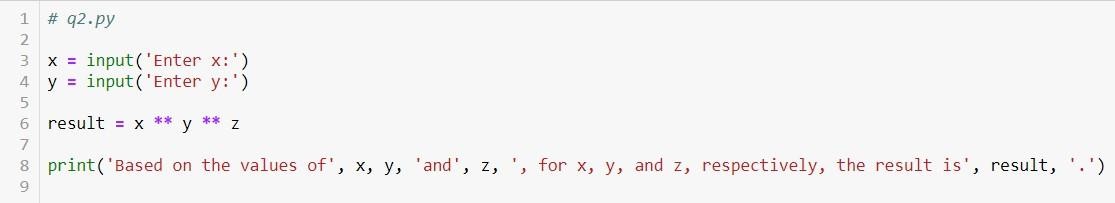
Text

Description automatically generated

## 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>

x = int(input(‘Enter x:’))

y = int(input(‘Enter y:’))

z = int(input(‘Enter z:’))

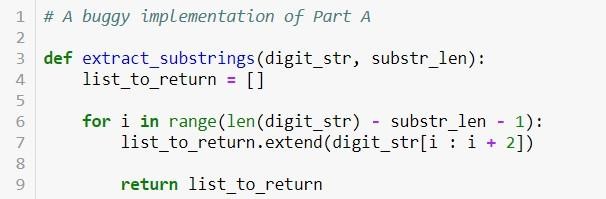
‘Based on the values of … no COMMA

## 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 [].

No gap

list\_to\_return.append(digit\_str[i:i+substr\_len])

for I in range(len(digit\_str) – substr\_len + 1):

Eg: I = [1], L = [2,3], i.extend(L) # I = [1,2,3]

### 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):

final\_list = []

for i in range(1, len(digit\_str)+1):

final\_list.extend(extract\_substrings(digit\_str, i))

sum = 0

for num in final\_list:

sum += int(num)

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.'],

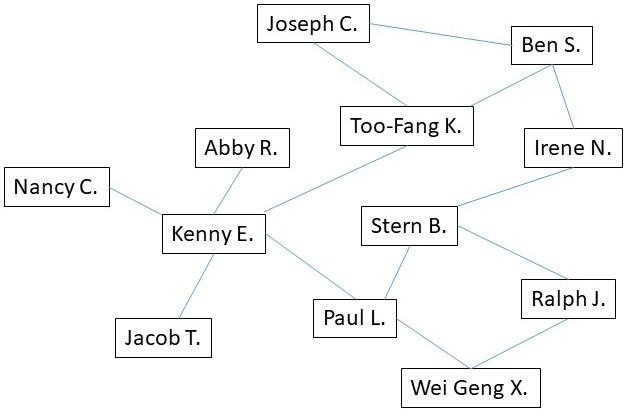
'Wei Geng X.': ['Paul L.', 'Ralph J.'],

'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):

# Create a list to return, with original people from case\_list

final\_list = case\_list.copy()

# For each degree of separation

for I in range(1, degree\_separation +1):

# Check each person inside final\_list, use copy() to not loop through a changing list

for person in final\_list.copy():

contacts = contact\_network[person]

# Append each person into the final\_list, if they are not already inside

for contact in contacts:

if contact not in final\_list:

final\_list.append(contact)

# Remove the original people in case\_list

for person in case\_list:

final\_list.remove(person)

# Returns an empty list if degree-separation == 0

return final\_list

## 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()

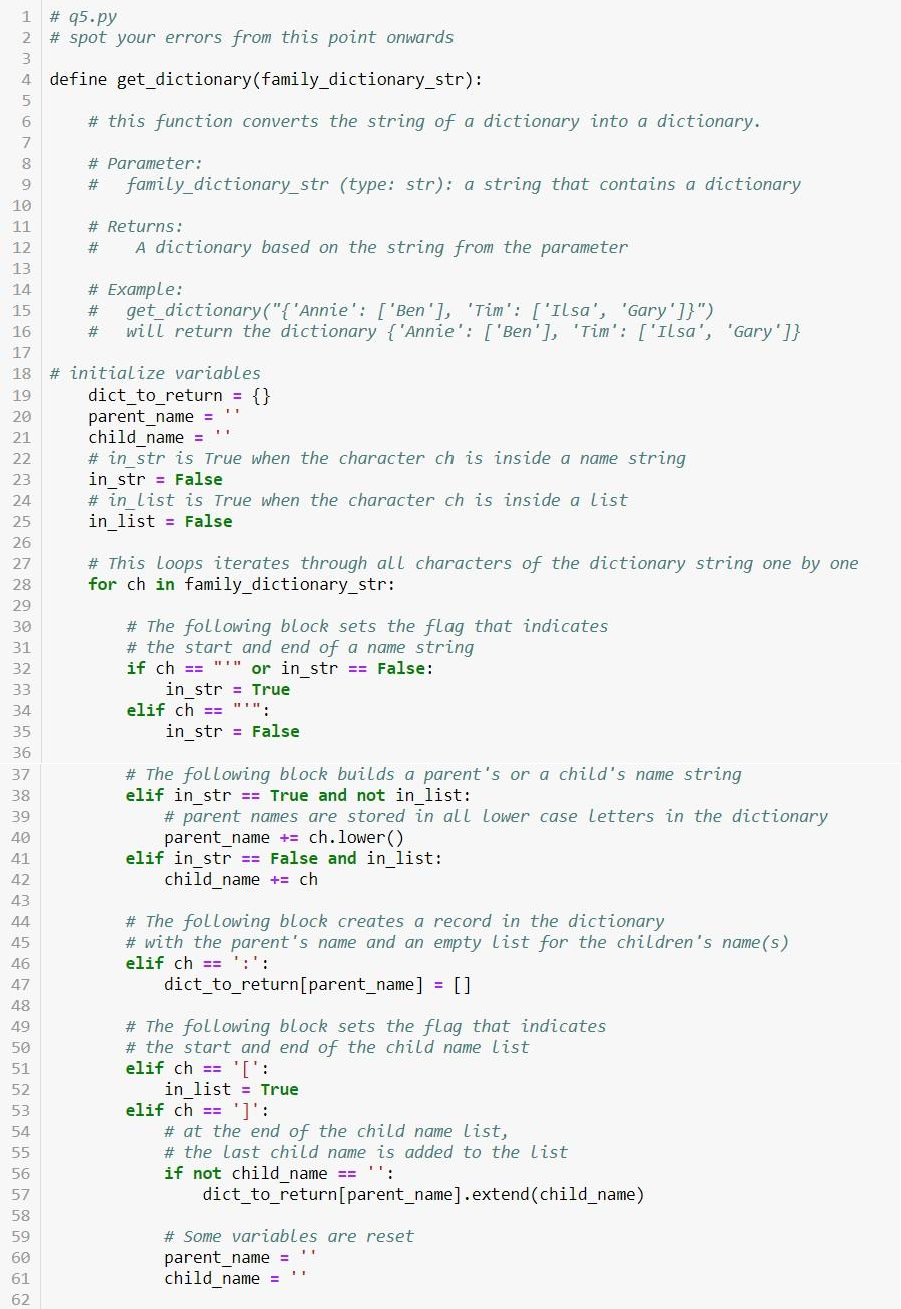
result = q5.lookup\_names(family\_dict\_str, 'Will') print('Test 3')

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()



If ch == “’” and not in\_str

def

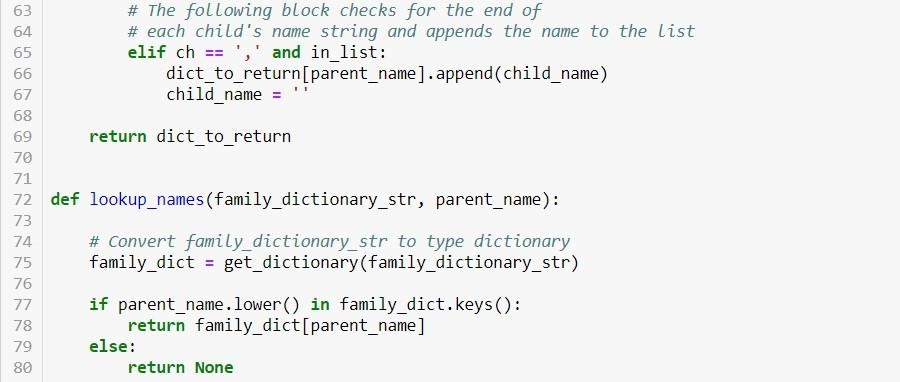
dict\_to\_return[parent\_name].append(child\_name)

In\_list = False

If ch == ‘:’:

elif in\_str and in\_list:

If in\_str and not in\_list:



return family\_dict[parent\_name.lower()]

## END OF PAPER.

**ENJOY YOUR HOLIDAY AND STAY SAFE!**