# Computational Thinking and Problem Solving (COMP1002) and Problem Solving Methodology in Information Technology (COMP1001)

Final Examination (19 December 2020)

Please read the instructions below carefully before you begin.

## General Rules

1. This is an open-book and open-notes examination, but closed-internet test (except for accessing BlackBoard).
2. The duration is 180 minutes.
3. No communication with others in any form is allowed. Violation could be considered as cheating / plagiarism.
4. Answer all questions.

## Submission Instructions

Follow the steps below:

1. Create a folder and name it as <student no>_<your name>, e.g., **`12345678d_CHANTaiMan`**
2. For Q1, Q2, Q3, Q4 and Q5, you need to submit the source file (**`.py`**). Name the **`.py`** files as Q<question no>_<student no>_<your name>**`.py`**, e.g., **`Q1_12345678d_CHANTaiMan.py`**
3. For Q6, you need to type your answers in a word document (include drawing as needed by scanning or photo-taking into the document) and save it as a **`.pdf`** file. Name the single **`.pdf`** file as <student no>_<your name>**`.pdf`**, e.g., **`12345678d_CHANTaiMan.pdf`**
4. Put all the **`.py`** and **`.pdf`** files into the folder.
5. Compress the folder (**`.zip`**, **`.7z`**, or **`.rar`**).
6. Submit the file to Blackboard.

You are allowed to submit **multiple times**. **Only the last attempt will be graded**.

***Note that NO late submission will be accepted. You will receive zero automatically.***

***Please submit your partially completed answers once in a while before the examination end time to avoid last minute rush and any "accident".***

1. [15 marks] (Input Data Validation) Suppose you need to write a program that checks the content type of an input. Given the following Python program template, complete the missing part of the program.

```
def main(inputString):

    if isInteger(inputString):
        print("The input is an integer.")
    elif isLetters(inputString):
        print("The input contains only English letters.")
    elif moreThanTwoSpaces(inputString):
        print("The input contains more than two whitespaces.")
    elif moreThanASingleLine(inputString):
        print("The input contains more than one line.")
    else:
        print("Unrecognized Input!")

def isInteger(s):
    # This function returns True if s is an integer.
    # Otherwise, return False. [4 marks]

def isLetters(s):
    # This function returns True if s contains only lowercase and uppercase
    # English letters.
    # Otherwise, return False. [4 marks]

def moreThanASingleLine(s):
    # This function returns True if s contains more than one line.
    # Otherwise, return False. [3 marks]

def moreThanTwoSpaces(s):
    # This function returns True if s contains more than two whitespace characters
    # Otherwise, return False. [4 marks]
```

Sample Testing Steps:

```
main("1001")
main("-1002")
main("abcdeABCDE")
main("Python is a programming language.")
main("Line 1\nLine 2")
main("code #123")
```

Sample Output:

```
The input is an integer.
The input is an integer.
The input contains only English letters.
The input contains more than two whitespaces.
The input contains more than one line.
Unrecognized Input!
```

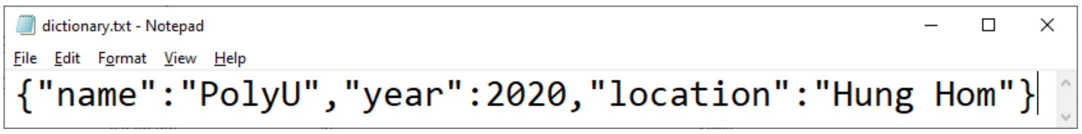
2. [15 marks] (Ticketing System) Suppose you need to implement a simple ticketing system. Given the following Python program template, complete the missing part of the program. Assume that both row and column numbers are correct and thus no data validation is needed. Note that the seat at the upper-left hand corner is (0, 0).

```
ROW = 5
COL = 5
seats = [[],[],[],[],[]]

def main():

    # Initialize the seating plan [2 marks]

    showPlan()

    sell(1, 2)
    sell(3, 4)
    sell(4, 1)
    sell(1, 3)
    sell(4, 1)
    cancel(2, 1)
    cancel(3, 4)

    print()

    showPlan()
    print("Available Seats:", availableSeats())

def showPlan():
    # This function displays the seating plan on the screen. [3 marks]

def sell(row, col):
    # This function sets the seat to 'x', meaning that it is sold.
    # If the seat can be sold, print "(row, col) sold successfully." on the screen.
    # If the seat has been sold already, print "(row, col) has been sold." on the screen.
    # [4 marks]

def cancel(row, col):
    # This function cancels the ticket and restores the seat.
    # If the seat has not sold yet, print "The seat has not been sold yet." on the screen.
    # [4 marks]

def availableSeats():
    # The function returns the number of available seats. [2 marks]

main()
```

The output of the program must look like below:

```
   Stage

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

(1, 2) sold successfully.
(3, 4) sold successfully.
(4, 1) sold successfully.
(1, 3) sold successfully.
(4, 1) has been sold.
The seat has not been sold yet.

   Stage

* * * * *
* * x x *
* * * * *
* * * * *
* x * * *

Available Seats: 22
```

3. [20 marks] (File and Data Conversion) Write a Python program that first reads a text file, named **`dictionary.txt`**, that contains a string representation of a Python "dictionary". For example,

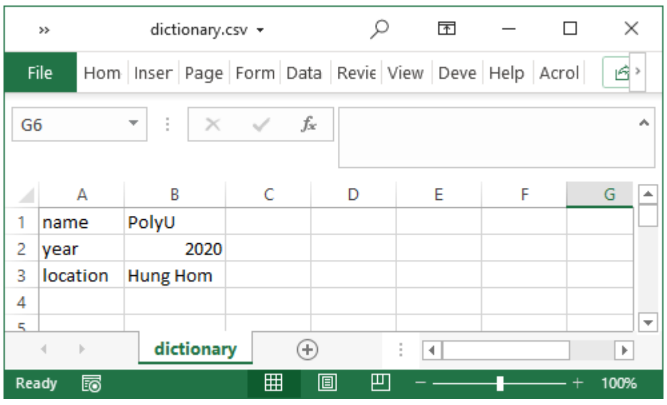


```
{"name":"PolyU","year":2020,"location":"Hung Hom"}
```

The string starts with "{" and ends with "}". Inside the brackets, it has a set of key-value pairs, in which the types of key and values can be **`string`** or **`integer`** only. Pairs are separated by commas. A string must be enclosed by "double" quotation marks and an integer has no quotation marks. Assume that the data in the text file is correct in format and no validation is needed.

Extract the key-value pair to a dictionary variable in your Python program. Print the dictionary variable to the screen. The keys and values must be the same type as those specified in the text file. Using our example, 2020 is an integer while the rest are strings.

Read the data from the dictionary in your program and write the data to a **`.csv`** file, named **`dictionary.csv`**. When the file is opened in a spreadsheet software, it should look like below:



| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | name | PolyU | | | | | |
| 2 | year | 2020 | | | | | |
| 3 | location | Hung Hom | | | | | |
| 4 | | | | | | | |

For this question, you can only use the functions/libraries that have been covered in the course, and the **`eval()`** function is NOT allowed.

4. [20 marks] (Soldier Counting) There was an old problem to count the number of soldiers *N* in a platoon. One may arrange the *N* soldiers in 3 rows, and there may be someone left behind. Similarly, arranging them in 5 rows, and in 7 rows, there may also be someone left behind. Take for an example, if *N* = 52, arranging them in 3 rows will lead to 17 per row, with 1 left behind. Arranging them in 5 rows will lead to 10 per row, with 2 left behind and arranging them in 7 rows will lead to 7 per row, with 3 left behind. The problem is to find the number of soldiers *N* in the platoon when given these three remaining counts of soldiers and in this example, 1, 2 and 3 respectively. If there is more than one possible answer, take the smallest one. In this case, you should return *N* = 52. Members in Kero's platoon are trying to solve this problem.

   (a) Kero considers a naive solution to start trying from *N* = 0, 1, 2, 3, … to see if the number of left-behind soldiers match, until the correct *N* is found. In this case that the number of left-behind soldiers are 1, 2 and 3 respectively, you will find that *N* = 52 and can verify that it is correct: 52 soldiers in 3 rows, 5 rows and 7 rows will have 1, 2, 3 soldiers left behind respectively. Write the pseudo-code for this naive solution. Provide also the input and output section in your pseudo-code. *Put your answer as a docstring in your Python program for this question at the top and submit it together with other parts of Q4 in one* **`.py`** *file*. [5 marks]

   (b) Kero instructs Doro to implement a Python function **`naive(r3,r5,r7)`**, to compute and return *N*, based on the pseudo-code he proposes. The parameters are the remainders when dividing *N* by 3, 5, 7, e.g. **`naive(1,2,3)`**. Write the Python function **`naive(r3,r5,r7)`** for Doro. Give appropriate comments. [5 marks]

   ```
   >>> print(naive(1,2,3))
   52
   >>> print(naive(1,3,5))
   103
   ```

   (c) This problem had actually been solved more than two thousand years ago. Here is a "translated" poem that describes a very nice solution:

      *Three rare-70-year old people walk together; five plum trees bear 21 blossoms; seven intellectuals reunion for half a month (15 days); divide by one hundred and five to find out.*

   This simply means that you would find *N* by multiplying the remainder with 3 rows by 70, multiplying the remainder with 5 rows by 21, multiplying the remainder with 7 rows by 15, and adding them up. If the number is 105 or more, subtract 105 until you get $0 \le N < 105$. Therefore *N* = 52 for the case above. Write a Python function **`clever(r3,r5,r7)`**, given the remainders with 3, 5, 7 rows, to compute and return *N* based on this clever ancient method. Give appropriate comments. [5 marks]

   ```
   >>> print(clever(1,2,3))
   52
   >>> print(clever(1,3,5))
   103
   ```

   (d) Kulu always comes up with something surprising. He hates performing division and proceeds to generate three Python sets: **`s31`**, **`s52`** and **`s73`** ***without using division***. The elements of **`s31`** are numbers that when divided by 3 yield a remainder of 1 (e.g. 1, 4, 7, …). The elements of **`s52`** are numbers that when divided by 5 yield a remainder of 2 (e.g. 2, 7, 12, …). The elements of **`s73`** are numbers that when divided by 7 yield a remainder of 3. The answer will then naturally be the element(s) in the *intersection* of the three sets. Write a Python function **`nodivision(r3,r5,r7)`** to generate the three sets and return *N* as the chosen element in the intersection of the three sets. [5 marks]

   ```
   >>> print(nodivision(1,2,3))
   52
   >>> print(nodivision(1,3,5))
   103
   ```

5. [15 marks] (Primitive Photoshop) It is a common need in computing to process images or pictures. In the old days, pictures were presented as a matrix of symbols, with different symbols giving different visual perceptions. Let us consider a picture of height *h* and width *w*. There are only 5 levels of intensity: 0, 1, 2, 3, 4, represented as " ", "`.`", "`-`", "`+`", "`#`" respectively, with increasing "density". The picture is represented as a matrix of numbers in the computer, but will be printed as a picture to a user. The matrix will be read as input in a *row major order* from a list of numbers. There are two possible actions to be performed on the picture: "`I`" means to increase intensity, "`D`" means to decrease intensity. It is not possible to go over 4 for an increase and below 0 for a decrease. Here are some pictures after performing the specific actions (with *h* = 4, *w* = 3). Note that it may not be always true that D(I(P)) = I(D(P)) = P, since the levels of intensity cannot go beyond 0 and 4.

```
Input picture
.-.
-+-
+#+
# #
After action I
-+-
+#+
###
#.#
After action D
.-.
-+-
+++
+ +
After action I
-+-
+#+
###
#.#
```

(a) Write a Python function **`process(picList,height,width,actionList)`** to take a *list of integers* representing the picture, the *height* and *width* of the picture, followed by a *list of actions* ("`I`" or "`D`") to be performed. The function will print the input picture and subsequent pictures after carrying out the actions. You are NOT allowed to use any module nor **`eval()`** but can use standard Python data structures. You can assume that there is *no error in the input*. Give appropriate comments. [8 marks]
Sample execution:
```
process([1,2,1,2,3,2,3,4,3,4,0,4],4,3,["I","D","I"])
```
Sample output:
As shown above.

(b) It would be a challenge in not using any **if-statement** in a Python program, as with our pseudo-code implementation of adding large numbers. You are here to re-implement your Python function **`process()`** as **`process2()`** so that **`process2()`** does NOT use any **if-statement**. Similarly, you are NOT allowed to use any module nor **`eval()`** but can use standard Python data structures. Give appropriate comments. [7 marks]

6. [15 marks] (Graph Application) Consider the Pouring Water Puzzle. We show in Assignment 5 Q4(*b*) using a graph that we are able to measure out size of 3 *l*, 6 *l*, 9 *l* and 12 *l* using two bowls of size 9-*l* and 12-*l*, but unable to measure out a size of 4 *l*.

   (a) Draw a graph using two bowls of size 6-*l* and 8-*l* respectively to show that we are able to measure out size of 2 *l*, 4 *l*, 6 *l* and 8 *l*. For simplicity, you do not need to draw the edges going back to the four basic states, i.e. (0 0), (6 0), (0 8), (6 8). [5 marks]

   (b) It is clear that we are unable to measure out size of 1 *l*, 3 *l*, 5 *l* and 7 *l* using two bowls of size 6-l and 8-*l*. Assume now that we are able to mark the ***middle level*** of the 6-*l* bowl. In other words, we are now able to measure out **exactly 3 *l*** from the 6-*l* bowl by filling it up to the **half-bowl mark** or emptying it down to **half-bowl mark**. We have four new operations related to this half-filled bowl: **F3**, **E3**, **P38** and **P83**. They mean filling to half of the 6-*l* bowl, emptying to half of the 6-*l* bowl, pouring from the 6-*l* bowl to 8-*l* bowl until it is half full or 8-*l* bowl is full, and pouring from the 8-*l* bowl to 6-*l* bowl until it is half full or 8-*l* bowl is empty respectively. Draw a partial graph showing the necessary states within four steps from (0 0) to indicate that you can measure out of **any size except for 7 *l* within 4 steps**. To simplify the graph, you do not need to draw any backward going edges to some earlier states. Thus, all edges going back to the 4 basic states (0 0), (6 0), (0 8), (6 8) and the 2 potential additional basic states (3 0) and (3 8) are not required. Note that it is actually possible to measure out 7 *l* in 6 steps, but this harder problem of measuring out 7 *l* is not required in this question. [10 marks]