NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Nanyang Business School

Programming Essentials
2020 July (30%)

Group Project (3 students in a group)

<< Safe Entry Management >>

Background:

More and more places require a very dynamic and mobile solution to track their visitors and the duration of their stay within the building. Such implementation helps in contact tracing. A simple set up of such a solution is shown in figure 1.

Fig 1 : The simple set up of a contact tracing application

Project Specifications:

1. Each entrance and exit to the building are equipped with a notebook pc and a barcode scanner each.
2. The barcode scanner is used to scan the bar code on the visitor identity card. The scanner connects to a PC which will run your python program.

3. A python command line application is loaded in the notebook PCs with the following menu options:

    ```
    ***** Foot Print Contact Point *****
    D: Set Door Gate
    C:  Configure PC number
    R : Entrance & Exit Tracking
    M: Merge Input/ Output Files
    Q : Quit
    ```

    Since the notebook PCs are movable, the same application in a particular PC may be activated by the user multiple times a day, at different entrances or exits.

4. The features of the recoding of entrance and exit tracking (option R in 3) is as follows:
    a. Program must check a valid 1 or 2 characters door gate ID already set, if not, it will show "Please enter a valid door gate before recording visits, press any key to return to the menu." Then allow user to view the error message, press any key then exit to the menu. Door gate ID may be any alphanumeric characters – no punctuation mark allowed. Door gate ID shall be stored in a file named "ID-DoorGate.txt" with exactly and only the door gate ID characters in it.
    b. Program must check a valid integer PC number is assigned to the current notebook pc. If the pc number is not set, you are required to allow the setting on the spot by calling a common function refined for (5) below before processing to (c). PC number shall be stored in a file named "ID-PCNumber.txt" with exactly and only the PC number digits in it.
    c. Allow user to scan in the ID (may be simulated by keyboard entry)
    d. Enter "e" for "Entrance (IN)" or "x" for "Exit (OT)" mode, or "Q" (capital Q) for "Quitting" the program.
    e. If it is an entrance mode, enter the contact number.
    f. Capture the data into the relevant csv file as stated in (7) below
    g. Repeat the process from scanning ID to capturing data into relevant csv file, until the computer detected a "Q" which terminates the loop and returns to the main menu.

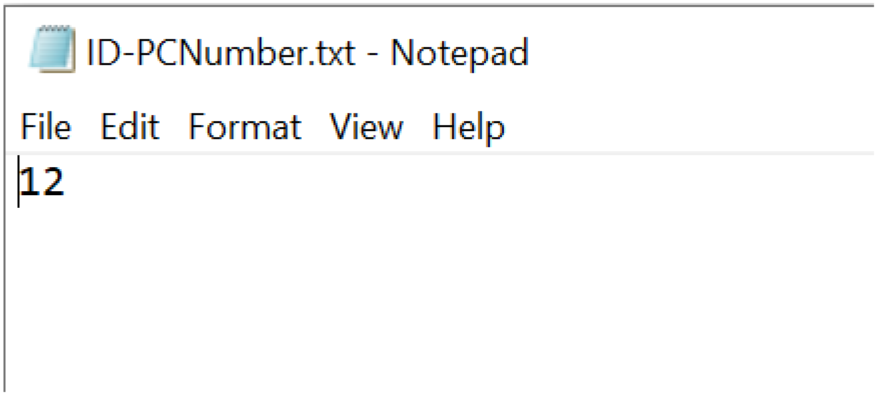
5. The features of Configure PC number (option C in 3) is as follows:
    a. A reusable function call acceptInteger1To99 is to be defined to ensure that program can only accept integer input from users from 1 to 99. This function should accept 2 parameters, question and error (as shown, you are required to follow the syntax here). The function will prompt the user with the "question" argument. When the user enters any invalid entry, the screen will show "error" argument. The pre-determined syntax of this function is as follows:

    ```
    def   acceptInteger1To99(question, error):
    ```

    b. The program will call the function acceptInteger1To99 with the following code (you are not supposed to change the way this is called)

    ```
    PCno = acceptInteger1To99("Please enter PC Number (1
    to 99)", "Invalid entry, please enter any number
    from 1 to 99 only")
    ```

    c. The program will use PCno as the global variable in the entire application
    d. PC number shall be stored in a file named "ID-PCNumber.txt" with exactly and only the PC number digits in it.

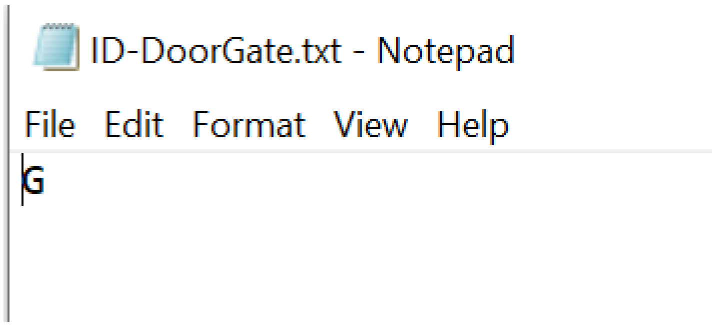


```
12
```

6. The features of Set Door Gate (option D in 3) is as follows:
    a. The user will be prompted to enter the door gate ID
    b. The program must show "Please enter a valid Door Gate ID (not more than 2 characters) if input is blank or more than 2 characters.
    c. The program will continue to prompt for input until a valid input is obtained.
    d. The program will use gateID as the global variable in the entire application.

e. Door gate ID shall be stored in a file named “ID-DoorGate.txt” with exactly and only the door gate ID characters in it.

Sample content in “ID-DoorGate.txt”



G

7. All the data collected at the gates are captured into 2 types of csv files. The file name formats will be fixed as

   IN_[YYYYMMDD]_[GATE]_[PC]_[startTime].csv

   for all entrance to the building and

   OT_[YYYYMMDD]_[GATE]_[PC]_[startTime].csv

   for exiting from the building. For both IN and OUT files, [startTime] must be rounded down to nearest whole hour. So for example, if gate crossing occurs at 14:36 hr, the data should be written to the file with [startTime] as 1400. The csv file format layouts are shown below:

   a. IN_YYYYMMDD_[GATE]_[PC]_[startTime].csv

      e.g. IN_20200504_A_01_1400.csv records all visitors who entered via gate A using notebook PC 01 from 2 pm onwards. Data Format in spreadsheet view is as follows :

Date	Time	Gate	PC	NRIC	ContactNo
2020-05-04	14:06	A	01	S9876554B	98765441
2020-05-04	14:08	A	01	T0176575J	87878773

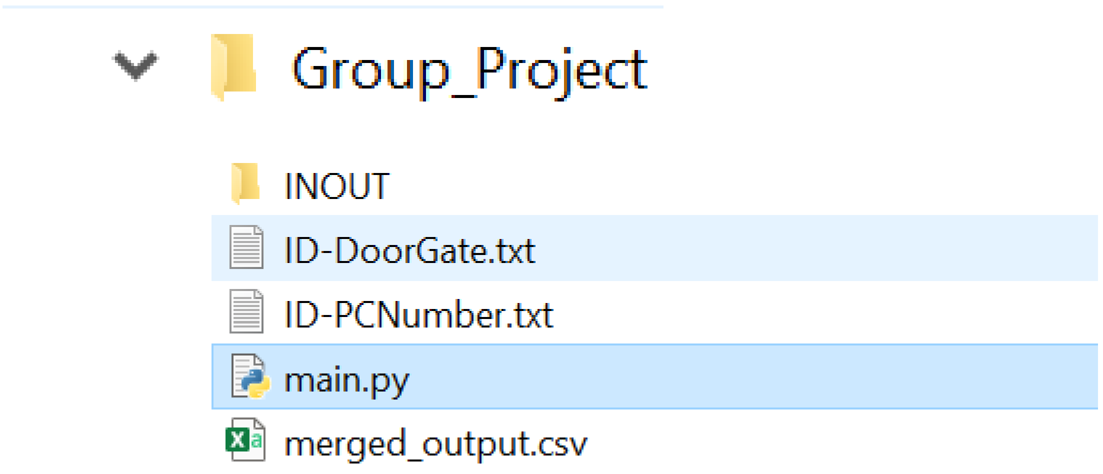
b. OT_YYYYMMDD_[GATE]_[PC]_[startTime].csv

e.g. OT_20200504_D_12_1400.csv records all visitors who entered via gate D using notebook PC 12 from 2 pm onwards. Data Format in spreadsheet view is as follows :

Date	Time	Gate	PC	NRIC
2020-05-04	14:45	D	12	T0176575J
2020-05-04	15:30	D	12	S9876554B

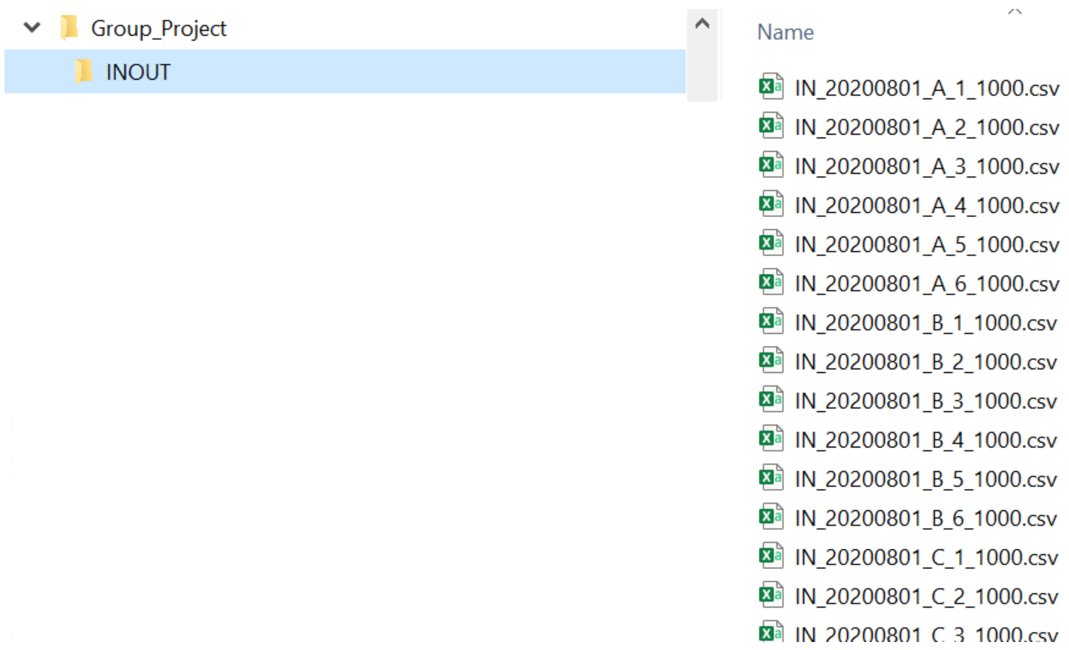
8. All .csv files will be written to a subdirectory “INOUT” under your project home directory (as shown in the diagram below). CSV files stored in any other place will NOT be marked nor considered correct. If “INOUT” subdirectory was initially not there, your program must create it automatically.

You program and configuration files should appear as follows:



Group_Project
INOUT
ID-DoorGate.txt
ID-PCNumber.txt
main.py
merged_output.csv

Within the INOUT sub folder you should have :



IN_20200801_A_1_1000.csv
IN_20200801_A_2_1000.csv
IN_20200801_A_3_1000.csv
IN_20200801_A_4_1000.csv
IN_20200801_A_5_1000.csv
IN_20200801_A_6_1000.csv
IN_20200801_B_1_1000.csv
IN_20200801_B_2_1000.csv
IN_20200801_B_3_1000.csv
IN_20200801_B_4_1000.csv
IN_20200801_B_5_1000.csv
IN_20200801_B_6_1000.csv
IN_20200801_C_1_1000.csv
IN_20200801_C_2_1000.csv
IN_20200801_C_3_1000.csv

9. The “Merge” option in the menu will do a data cleaning to the collected data and deposit the cleaned data into a consolidated csv file “merged_output.csv”. The format of the consolidated file is as follows:

Date	In Time	In Gate	In PC	ContactNo	Out Gate	Out Time	OutPC	StayMinsDuration
2020-05-04	14:06	A	01	98765441	D	15:30	12	84
2020-05-04	14:08	A	01	87878773	D	14:45	12	37

Note that

a. you are given a zip that contains one month’s collections of entrance and exit data, you are required to combine them into 1 file as shown above.
b. The IN and OUT data files are supplied in the INOUT\ subdirectory in your project home directory.
c. Your merged output CSV file should also be stored in your project home directory.
d. there is an extra column call StayMinsDuration, this is the duration the visitor spent in the building in minutes, it is the difference in InTime and OutTime. Your program should take into account differences in dates as well.

<u>Submission Procedure:</u>

Your group must include in a submission zipped file ALL project programs, presentation slides, etc stored in the specified subdirectories. This submission zipped file MUST be a single file named as:

AN6100-Cx-Ggg-Project.zip

where x is A or B depending on whether you are in class A or B, and gg is a 0-filled group number (eg group 1 will use 01 to replace gg).

To submit, ONE and only one member from your group will send email to your instructor's email AND cc to NTU emails of ALL members of group by (1) attaching the above submission zipped file, and (2) using this same filename as the only subject line. The email itself need not have content (unless you want to say hello).

Do note that:
(1) The submission zipped file must not be too large or else your email may not be delivered.
(2) Deadline for submission is 20 Sep 2020 (Sunday) 12:00pm. This is a strict deadline - no extension for any reason will be permitted.
(3) Submission MUST be received by your instructor BEFORE the deadline. Eg, if you send your email, but your instructor receives it after the deadline, your group will be considered late.
(4) Your email MUST include ALL your group mate's NTU emails - failure to do so means your email is NOT a project submission email and will be considered as not having submitted.
(5) No other means of submission will be acceptable - do not invent new ways to send your zip file, like using third-party storage, etc. Instructors will not create new account or make special downloads just because your group stores it in a special location.
(6) No outside file beyond those in the submission zipped file will be graded. Eg, if after deadline, you send a sorry email trying to additionally add more files to your group's files, we will also be sorry about it and nothing will be accepted for grading.

General Rules and Guidelines on Programming:

1. ALL files must be stored in the same directory. Unless otherwise specified, no sub-directory may be created before or during program run. All files include programs, text files, database files, temporary files, etc.

2. Your python scripts must be platform independent. They will be tested on Windows-based machines.

3. Submission of all files must be done as a zipped file using ".zip" as the archiving format. When unzipped, no directory must be created – all individual files will be directly extracted.

4. Each submitted project MUST have a file named "main.py" which will be the first program being run. If a project does not have such a file, the project will get 0 mark.

5. Within the "main.py" module, there must be a function defined as:

```
def main():
```

which will be the first function being called to run your project. If no `main()` function is present, the project will get 0 mark.

6. Project may import other modules, including standard modules and your own derived modules which must be available readily when the submitted zip archive is unzipped. standard modules consist of modules mentioned during the course, such as during seminar, assignment, reading list, etc (where applicable). Do not assume that the assessment environment will always have any module other than standard modules. If you try to load a module which the assessment environment does not have, your project will suffer mark penalty due to malfunction or inability to even start.

7. The total execution time of your submitted project is 5 minutes (of computer execution time), excluding user input response time. For example, if a calculation loop in your project takes 7 minutes to come to a result for display, your project will be force-terminated.

8. Unless otherwise stated, all user inputs will be captured by your program through the standard input channel. One such function to capture standard input channel will be `input()`. Unless otherwise stated, all output printouts will be sent to the standard output channel. One such function to send textual outputs to standard output channel will be

`print()`. This means that if your program works but requires inputs to be delivered via other channels, or your program sends output to other channels, no mark will be given.

9. All files, including submitted files in zipped archive and program-created temporary and permanent files, must be named according to the format given in the relevant description of project guideline or related documentation. If your project has the correct file content but is named incorrectly or not strictly according to given format or template or cannot be opened by the testing system, your file may be skipped from grading, be treated as missing file, and not be awarded any mark.

10. No re-grading, appeal or re-assessment of any sort will be conducted. All grading and assessments will be done on and only on the submitted zipped archive of files. No separate delivery of additional or replacement file, via email to instructor or otherwise, will be accepted.

11. Late submission of project will be given 0 mark.

12. Project grading and deadline are final. Please ensure that your project work adheres to all required guidelines punctually, rather than plead for leniency or special treatment after the deadline.

13. No scores will be announced after the grading, should there be any issues with the coding, only qualitative feedback will be given.

<u>Project Presentation:</u>

Each group will present your project work, regardless of whether your python programs run successfully or otherwise. Though your group will be presenting to the class and instructor, you should assume that the audience is the Management Board deciding whether your proposed project will be approved for funding and execution or not.

Your entire presentation content is to convince the Board that your team is capable and qualified to deliver the requirements, by illustrating your business and programming mastery of the project. Thus, while demonstrating your working code is great, it would be the overall presentation persuasiveness that would be graded. Try to keep this question in your mind, "Why should the Management Board appoint your team and not other competing teams to take responsibility of this project?"

As the program specification is quite tight, it is not necessary to re-describe the program and/or file specifications again. Instead, the context and surrounding aspects pertaining to such a project, if done in real life, will be interesting points for discussion. Here are some *example* aspects for your consideration:

a. Objectives, purpose, and justification of having this project implemented
b. Estimates of volume of traffic served
c. Beneficiary stakeholders if the project is run, or damaging stakeholders if not run
d. Assumptions made
e. Budget required
f. Operational details when project is deployed
g. Sample outputs and analysis

Note that these are just examples – covering all these does not mean your presentation is good, nor using other aspects not mentioned above would be considered poor.

Again, to re-emphasize, it is the overall persuasiveness of your group's presentation, probably coupled with short but convincing live demo, that would be getting the best scores.

<u>Assessment Rubric:</u>

Your project must work! The bulk of the assessment will be on whether your program delivers the kind of required outcomes. Both on-screen textual outputs and off-screen file outputs will be assessed for correctness. In addition, your program codes will be assessed for execution, clarity of design, code readability and documentation decency.

	BELOW EXPECTATION	MET EXPECTATION	ABOVE EXPECTATION
Execution (20%)	Program generally fails to work. While parts of program may be working, the entire program does not achieve the required goals. Program may deliver faults, halts	Program generally works with some test scenarios, though not all outputs are correct. No mid-way termination occurs with most test	Program works very well in almost all cases with correct expected outputs. Program handles proper-input scenarios and unexpected nasty input scenarios graciously

	mid-way, or times-out by taking extremely long time for execution.	scenarios, though program may fail when unexpected inputs are sent, or under stress test scenarios.	without crashing. Within limits of the test cases, program is thought to be functioning perfectly.
Coding (20%)	Codes do not seem to terminate properly, will cause syntax error or cannot load at all.	Codes are syntactically clear, able to load for execution, and do not seem to have visible errors, though some parts may still seem rather convoluted or lengthy.	Codes are syntactically clear, able to load for execution and visibly clean. Such standard appears to be maintained across all program modules.
Reusability (10%)	Little to no use of proper segmenting of program logic, resulting in one-huge-program design. Excessive modularization or functionalization such that clarity of logic is lost.	Some usage of modularity and functions to factorize execution logic, though some parts still exhibit one-large-program phenomenon. Or, somewhat excessive use of modularization or functionalization, resulting in some parts being choppy, one-liner.	Visibly appropriate usage of modularity and functions to deliver clarity of logic and program execution. Such standard appears to be permeated across all program modules.
Maintainability (10%)	Code fragments are confusing, lengthy, bulky, do not convey logical clarify, hard-to-understand, etc	Some level of usage of clear labels for variables and functions to provide ease of reading the program codes. Somewhat suitable amounts of code for conditionals and loops, though lengthy codes still are visible.	Very good use of readable labels for variables and functions to provide consistent and standardized naming. Very good segmenting of coding lines within conditionals and loops to give clarity to logic. Such standard appears to be permeated across all program modules.
Documentation (10%)	Little to no documentation of program, functions, statements, etc. Makes maintenance of code difficult. Lacks ability for code to be shared.	Some amounts of documentation of functions and/or variables and/or key statements, although more would be expected. Where there's	Very good amount of documentation and comment of functions, variables and/or key statements. Makes code sharing very easy to other programmers. Makes maintenance

		documentation, the level of comment is shallow and cursory. Code may be sharable through such documentation, but may challenge maintenance efforts.	highly possible as the documentation of usage, intent, general logical goal is sufficient.
Presentation Content (10%)	Does not contain central message or identifiable organizational pattern.	Central message is not clearly and/or easily identifiable by audience; sections may be in need of further organization and clarity.	Contains a clear central message and clearly-identifiable sections featuring purposeful organizational pattern (e.g., chronological, problem-solution, analysis of parts, etc.)
Visual Aids (10%)	Typos throughout slides and; too much text on slides. Material on slides either is identical to speaker's speech or completely disconnected from it.	Slides with occasional typos, unclear organization, and/or questionable applicability to presentation. All slides include significant amounts of text.	Slides are professional and easy to read. Materials enable speakers to focus on presentation.
Presentation Skill(10%)	Members avoid or discourage active audience participation.	Multiple members went off topic and lost audience. Failed to utilize method to pull the audience into the speech.	Involved audience in presentation; held their attention throughout by getting them actively involved in the speech and using original, clever, creative approach.