

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ1003: INTRO TO COMPUTATIONAL THINKING

Real-time Canteen Information System

FS 12 GROUP 5:

Lam Jing Xuan Denise U1922510E
Lee Yi Hui Rachel U1922672E
Le Quang Anh U1922940J

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY**

Table of Contents (word count: 1200, excluding tables, figure labels, and code)

1. Flow of Programme	4
1.1. Flowchart	4
1.2. Code structure	5
1.2. Data files	7
2. Notable parts of the code	8
2.1 Function: enterDate()	8
2.2. Determining the status of stall: breakfastOrLunchOrDinnerOrClosed()	10
2.3 Displaying stalls	11
2.4 Function to determine waiting time: getWaitTime()	13
3. Error Handling Test Cases	14
4. Reflection	16
4.1. Group reflection	16
4.2. Individual reflection	17
5. Suggestions for Further Improvement	18
6. Work distribution	18

1. Flow of Programme

1.1. Flowchart

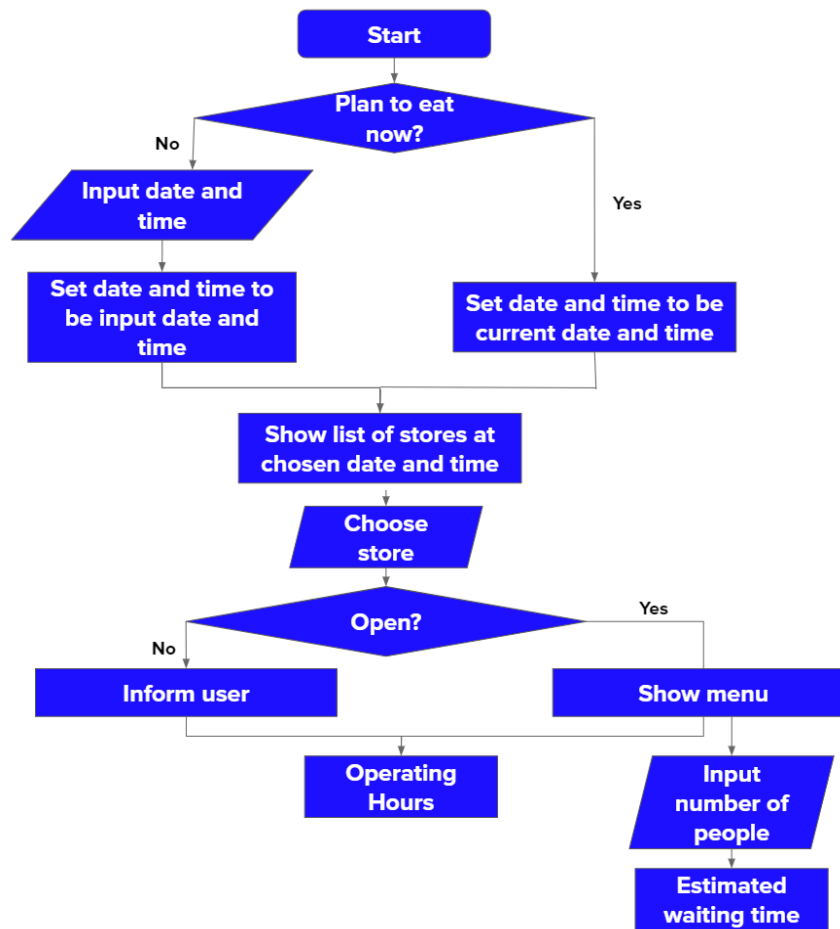
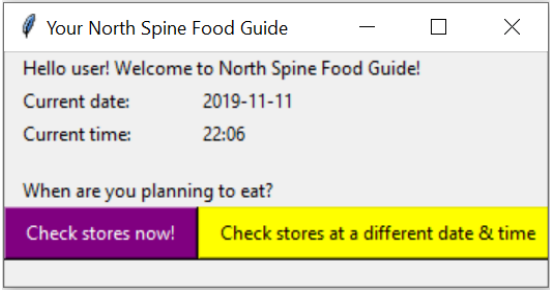
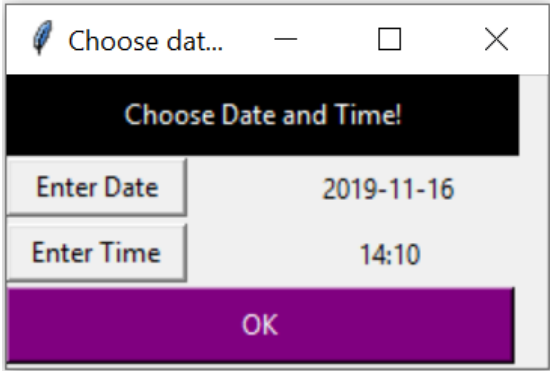
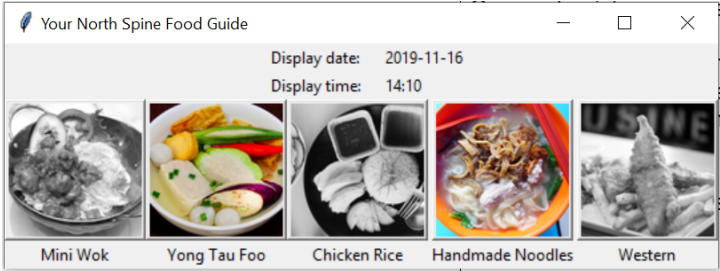
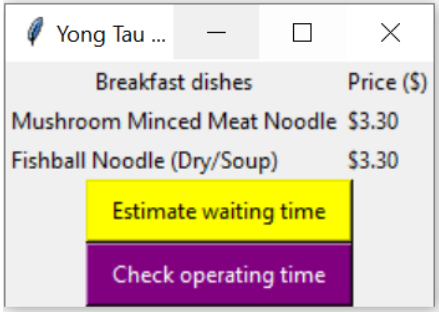
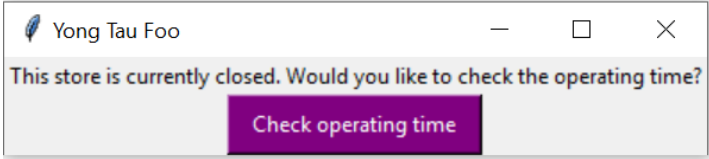
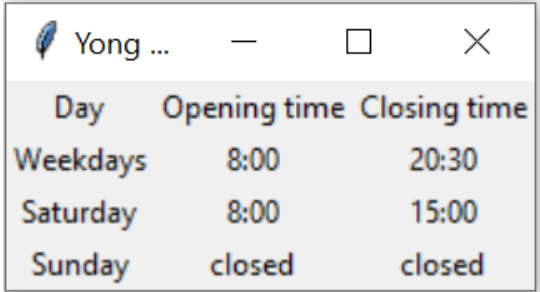
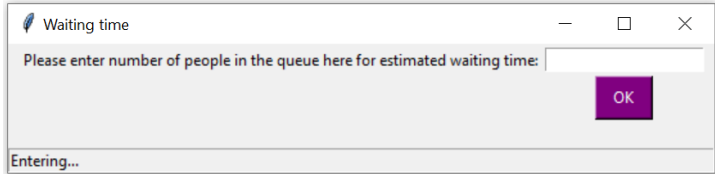


Figure 1: Flowchart

1.2. Code structure

File	Function
<i>FS12_Lee_Le_Lam.py</i>	Launches the application
<i>generateWindow_1.py</i> 	Main interface Displays current date and time “Check stalls now!” generates Window 3 “Check stalls at a different date & time” generates Window 2
<i>generateWindow_2.py</i> 	Window 2 Allows user to change time and date
<i>generateWindow_3.py</i> 	Shows stalls available at chosen time and date Grayscale pictures: closed stalls. Coloured pictures: open stalls. When a picture is clicked, window 4 is generated.

<p><i>generateWindow_4.py</i></p>  	<p>Window 4</p> <p>If stall is open, display menu according to time (breakfast, lunch, dinner) and the price of food items.</p> <p>“Check operating time”: generates Window 5.</p> <p>“Estimated waiting time”: generates Window 6.</p> <p>If store is closed, shows “Check operating time”</p>
<p><i>generateWindow_5.py</i></p> 	<p>Window 5</p> <p>Informs users of the operating hours of the stall.</p>
<p><i>generateWindow_6.py</i></p> 	<p>Window 6</p> <p>Calculates estimated waiting time.</p> <p>If integer is entered, the estimated waiting time will be input multiplied by two.</p> <p>Informs user of the waiting time.</p>

Whenever the user wishes to return to a previous window, the “X” button on the top-right corner of the current window closes it.

1.2. Data files

File	Information
stallList.csv	Name of stalls and associated image files
stallMenu.csv	Menu (breakfast, lunch, dinner)
operatingHours.csv	Opening and closing time on Weekdays, Saturday and Sunday
.png files	Coloured and grayscale images associated with stalls

1.3. Important external libraries

Library	Use
tkinter	GUI
pandas	Access the data in .csv
datetime	Handle date and time
tkcalendar	Calendar to choose date

2. Notable parts of the code

2.1 Function: `enterDate()`

Generates window allowing users to choose dates from a calendar.

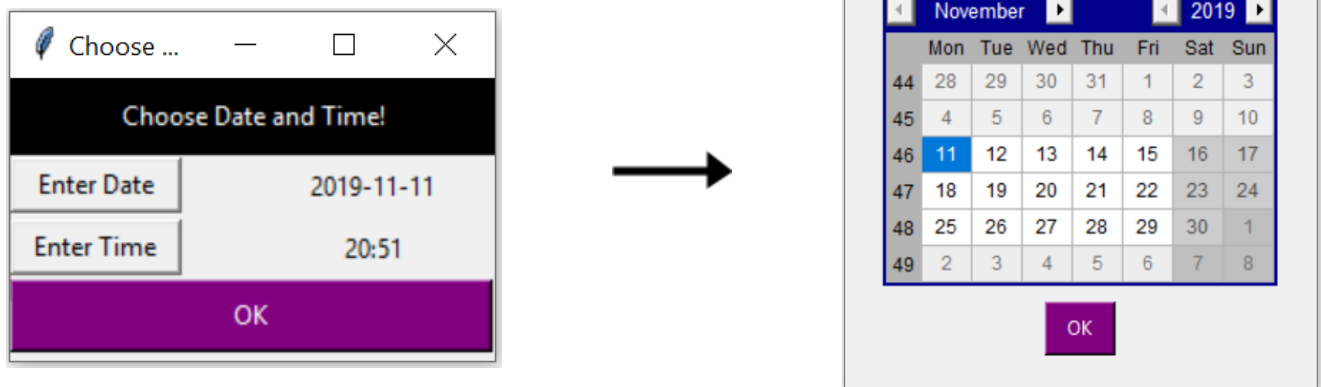


Figure 2: “Enter Date” button when clicked

The function `enterDate()` opens a top-level window (Figure 3, line 34). The calendar range is set from current date (line 42) to 365 days in the future (line 43). A `tkcalendar.Calendar` object is created with said parameters (line 45). Another function `getUserDate()` is defined to store the selected date in global variable `userDate` on the calendar (lines 49-53). An “OK” button is generated and when clicked, will activate `getUserDate()` (line 57). The selected date is stored only when the “OK” button is clicked, minimising the chances of users accidentally choosing a date while exploring the calendar.

```

31     def enterDate():
32         '''Let user set the date'''
33
34         top = Toplevel(window_2)
35         topText = 'Choose Date'
36         top.title(topText)
37         top.geometry('300x300+30+30')
38         chooseDateText = 'Please click to choose a date'
39         chooseDateLabel = Label(top, padx=10, pady=10, text=chooseDateText)
40         chooseDateLabel.pack()
41
42         mindate = currentDatePara
43         maxdate = mindate + datetime.timedelta(days=365)
44
45         cal = Calendar(top, mindate=mindate, maxdate=maxdate, width=12, background='darkblue',
46                        foreground='white', borderwidth=2)
47         cal.pack(padx=10, pady=10)
48
49     def getUserDate():
50         global userDate
51         userDate = cal.selection_get()
52         dateLabel['text']=userDate #update userDate in the dateLabel of window_2
53         top.destroy()
54
55         # end of getUserDate()
56
57     okDateButton = Button(top, padx=10, pady=5, bg='purple', fg='white', text='OK', command=getUserDate)
58     okDateButton.pack()

```

Figure 3: `enterDate()` code

2.2. Determining the status of stall: `breakfastOrLunchOrDinnerOrClosed()`

Breakfast, lunch and dinner menus of each stall differs depending on the time of the day. The menu will be unavailable if the stall is closed.

```
12 def breakfastOrLunchOrDinnerOrClosed(date, time, stallName, operatingHours):
13     '''Input:
14     1) date and time defined by user (datetime.date and datetime.time objects)
15     2) stallName is a string representing name of stall chosen by user
16     3) operatingHours is a dataframe of all the operating hours of the stalls, with stall name as index
17     Output:
18     1) Return 'Lunch' if it's lunchtime and 'Breakfast' and 'Dinner' and 'Closed' if closed
19     '''
```

Figure 4: determine status of stall

The opening and closing times are extracted from the dataframe `operatingHours` previously generated by `pandas.read_csv('operatingHours.csv')`. Operating times vary for Weekdays, Saturday and Sunday, so, we have to determine which day of the week is chosen (Figure 5, line 21-27).

```
21     dayOfWeek = date.weekday()
22     if dayOfWeek <= 4:
23         checkDate = 'Weekdays'
24     elif dayOfWeek == 5:
25         checkDate = 'Saturday'
26     else:
27         checkDate = 'Sunday'
28
29     openingTime = operatingHours.loc[stallName][operatingHours.loc[stallName, 'Day'] == checkDate]['Opening Time'][0]
30
31     if openingTime == 'closed':
32         return 'Closed'
33
34     openingTimeObject = datetime.datetime.combine(date, \
35         datetime.datetime.strptime(openingTime, '%H:%M').time())
36
37     closingTime = operatingHours.loc[stallName][operatingHours.loc[stallName, 'Day'] == checkDate]['Closing Time'][0]
38     closingTimeObject = datetime.datetime.combine(date, \
39         datetime.datetime.strptime(closingTime, '%H:%M').time())
```

Figure 5: opening and closing time

Next, if-else and comparisons are used to determine the status of the stall. Note that to compare two datetime objects, we need both the date and time. Hence, the `datetime.datetime.combine()` method combines date and time data (Figure 6, lines 41-45).

```
41     breakfastEndTimeObject = datetime.datetime.combine(date, datetime.time(11,0))
42
43     lunchEndTimeObject = datetime.datetime.combine(date, datetime.time(17,0))
44
45     userDefinedTime = datetime.datetime.combine(date, time)
46
47     if openingTimeObject > userDefinedTime or userDefinedTime > closingTimeObject:
48         return 'Closed'
49     elif userDefinedTime <= breakfastEndTimeObject:
50         return 'Breakfast'
51     elif userDefinedTime <= lunchEndTimeObject:
52         return 'Lunch'
53     else:
54         return 'Dinner'
```

Figure 6: comparison

2.3 Displaying stalls

Depending on date and time, operating stalls will be represented by coloured pictures and grayscale pictures if closed (Figure 7).

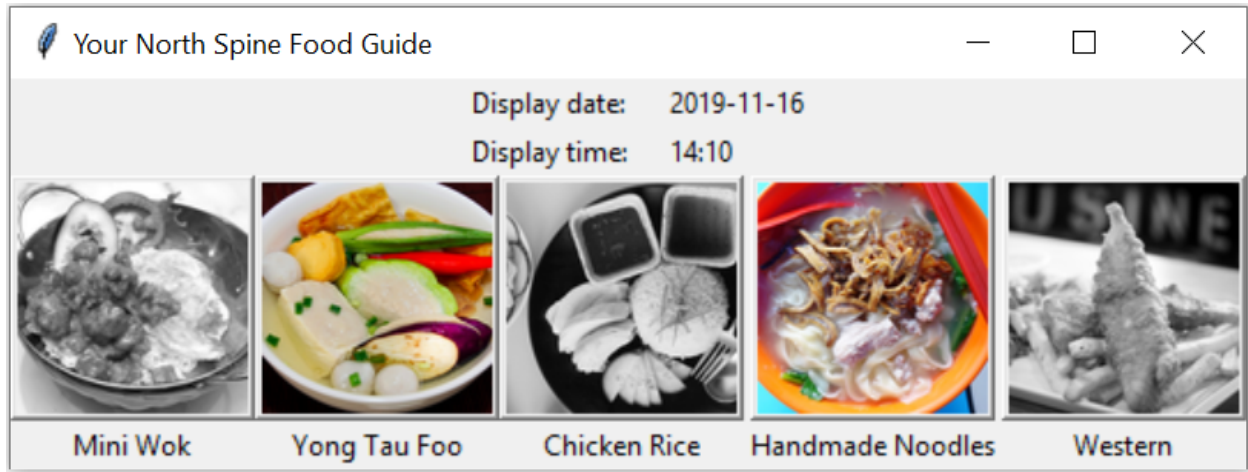


Figure 7: Stall display at chosen date and time

1	Stall	Cuisine	Location	Image	Greyscale Image
2	Mini Wok	Chinese	North Spine Food Court	mini_wok_pic.png	grey_mini_wok_pic.png
3	Yong Tau Foo	Chinese	North Spine Food Court	yong_tau_foo_pic.png	grey_yong_tau_foo_pic.png
4	Chicken Rice	Chinese	North Spine Food Court	chicken_rice_pic.png	grey_chicken_rice_pic.png
5	Handmade Noodles	Chinese	North Spine Food Court	handmade_noodles_pic.png	grey_handmade_noodles_pic.png
6	Western	Western	North Spine Food Court	western_food_pic.png	grey_western_food_pic.png

Figure 8: stallList.csv with the .png file names

A dictionary function was used to achieve this (Figure 9). A loop was formed to iterate through each row of data. The dictionary `ButtonDict[]` is used to generate a button and a label for each row. First, we determine whether the stall is open by calling `breakfastOrLunchOrDinnerOrClosed()` (line 96). Next, we made use of if-else statement to select the respective images according to the defined time. If store is open, we access the coloured image (line 100) and if closed, access the the grayscale image (line 103), by calling `PhotoImage()` with the file name obtained from `stallList.csv`. The images would then be displayed by passing it as a parameter for the `Button()` (line 107).

```

91     for i in range(numRow):
92         buttonDict[i] = [0,0,0]
93         row = df.iloc[i] # row Series
94         stallName = row[0] # name of stall
95
96         statusMealTime = breakfastOrLunchOrDinnerOrClosed(date=userDatePara, time=userTimePara, \
97                                                         stallName=stallName, operatingHours=operatingHours)
98
99         if statusMealTime != 'closed':
100             photo = PhotoImage(file=row[3])
101
102         else:
103             photo = PhotoImage(file=row[4])
104
105         buttonDict[i][1] = (lambda x=stallName, y=statusMealTime: stallButtonFunction(userDatePara=userDatePara, userTimePara=userTimePara, \
106                                                                 stallName=x, statusMealTime=y))
107         buttonDict[i][0] = Button(bottomFrame, image=photo, command=buttonDict[i][1])
108         buttonDict[i][0].image = photo
109         buttonDict[i][2] = Label(bottomFrame, text=row[0])
110
111         # row and column position for the stall buttons
112         rowPosition = 2 * (i // 5)
113         colPosition = i % 5
114
115         buttonDict[i][0].grid(row=rowPosition, column=colPosition)
116         buttonDict[i][2].grid(row=rowPosition+1, column=colPosition)

```

Figure 9: Code for displaying store

2.4 Function to determine waiting time: `getWaitTime()`

The user is prompted to input the number of people in the queue. Try-except statements were utilised to handle errors. Default waiting time is set at 2 minutes per person. After the user inputs a valid integer value, it would be multiplied by 2 and a messagebox indicating the waiting time will appear.

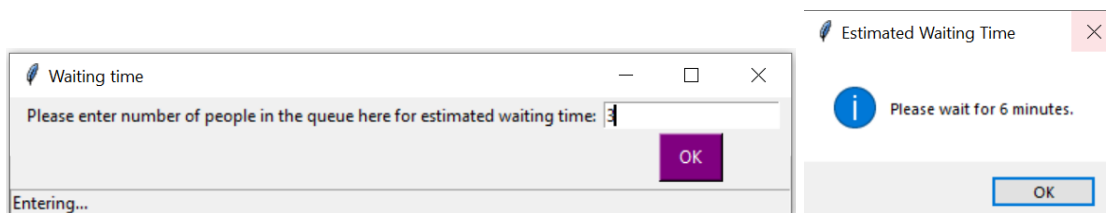


Figure 10: Messagebox indicating waiting time

3. Error Handling Test Cases

Errors may occur when user inputs (a) time in 24hr format and (b) number of people in queue. Possible error types are (i) non-integer input (ValueError) and (ii) integer input out of range. We implemented a status bar, with a function informing the user of invalid input. Value is only stored when a valid input is entered.

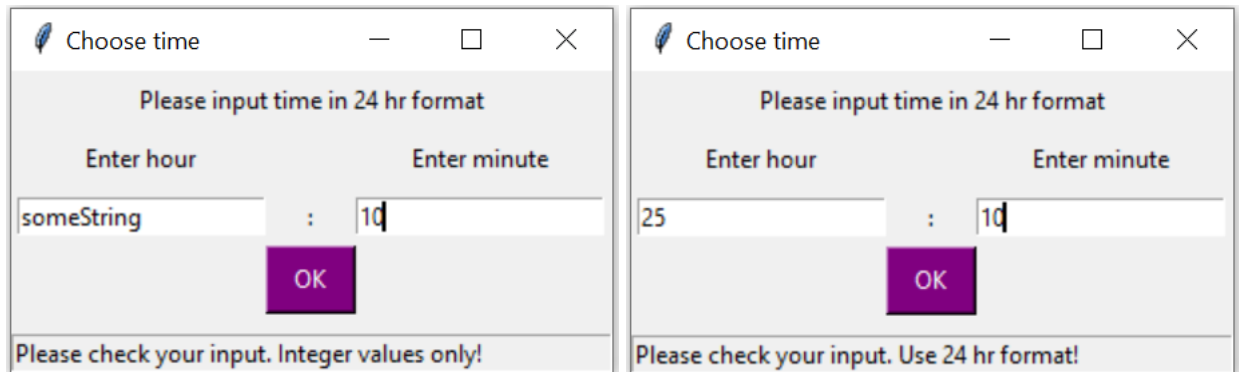


Figure 11: Clicking “OK” shows error messages

The “OK” button is bound to procedure `getUserTime()`, which stores input in global variable `userTime` (line 106) and closes the window (line 108) when valid input is entered (Figure 12). Otherwise, error messages will be shown in the status bar and the window remains for user to re-enter input.

```

97     def getUserTime():
98         hourEnteredStr = hourEntry.get()
99         minuteEnteredStr = minuteEntry.get()
100
101     try:
102         hourEnteredInt = int(hourEnteredStr)
103         minuteEnteredInt = int(minuteEnteredStr)
104         if 0 <= hourEnteredInt <= 23 and 0 <= minuteEnteredInt <= 59:
105             global userTime
106             userTime = datetime.time(hourEnteredInt, minuteEnteredInt)
107             timeLabel['text'] = userTime.strftime('%H:%M')
108             top.destroy()
109         else:
110             status['text'] = 'Please check your input. Use 24 hr format!'
111     except ValueError:
112         status['text'] = 'Please check your input. Integer values only!'
113
114     # end of getUserTime()

```

Figure 12: try-except clause

A similar code is used for cases (b)(i) and (b)(ii).

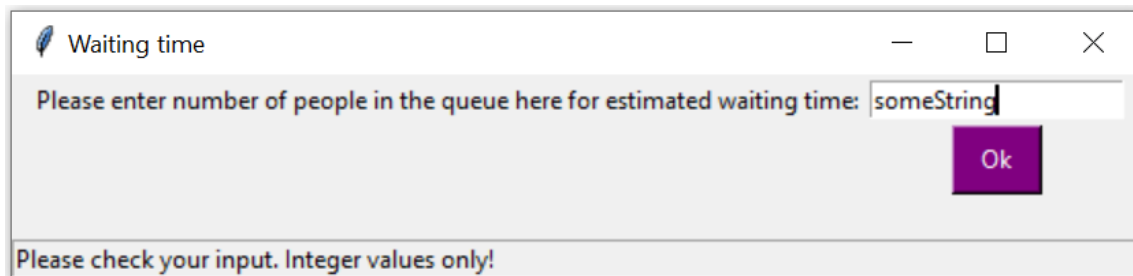


Figure 13: (b)(i)

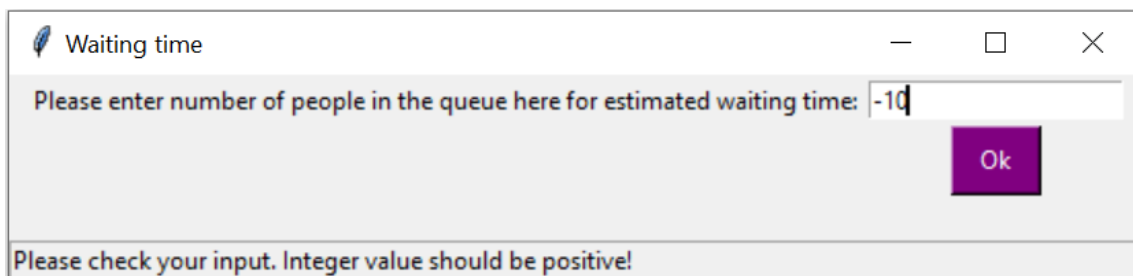


Figure 14: (b)(ii)

4. Reflection

4.1. Group reflection

To tackle this complex problem, we first tried to decompose it into smaller problems, with specific deliverables. The work was distributed among members: each person developed some specific functions for the app.

Initially, collaborating was difficult. Each person coded on our own laptops and sharing was cumbersome. We then discovered Github and henceforth utilised it. Each member was able to work on their allocated parts, share their code, and create pull requests for peer approval before contributing to the master branch, making our work much more efficient.

Debugging was challenging as we were unfamiliar with tkinter. Much time and effort was needed for extensive research to learn the various functions that we required. A notable bug occurred for the stalls' display pictures; the output differed every time we ran the code. We discovered that the problem lay with how tkinter operated. Adding a seemingly redundant line of code resolved the problem (Figure 15). It is crucial to understand the tool used, and to think of how the computer interprets the code.

```
107         buttonDict[i][0] = Button(bottomFrame, image=photo, command=buttonDict[i][1])
108         buttonDict[i][0].image = photo
```

Figure 15: line 108 is seemingly redundant

4.2. Individual reflection

Denise:

Learning how to utilise the various external libraries like tkinter and pandas was rather challenging for me. As a computer science student with little to no prior coding experience, understanding, much less designing a program flow was difficult. Even though it was mentioned in the group reflection, being resourceful was incredibly important. Since tkinter was not an external library that had been taught in CZ1003, I had to do extensive googling for function names and tutorials to have even have a rough idea on how to utilise it, especially with my then poor understanding of python and its syntax. This experience was incredibly helpful in teaching me how to properly utilise python, and has taught me far more than the simple lecture videos and LAMS sequences that we have been attempting. A truly humbling experience.

Rachel:

As I had no prior programming background, creating an app was a big challenge for me. I had trouble putting my thoughts into codes. Hence, I researched intensively on Google and watched tutorial videos to familiarise myself with the use of the programs. However, many of the information online are either outdated or incorrect. I learnt to extract only the relevant information. I also realised the importance of being conscientious when coding as minor errors would result in the program not being able to run. Although it got really frustrating after running into many problems, the various errors I encountered made me more aware of how programming actually works and what is accepted and not accepted. This knowledge would be really useful in the future and for other projects.

Anh:

Making the functions reusable for different scenarios was a crucial task. Initially the functions I coded contained few to no parameters since I thought it would be enough to solve the one task I was assigned. Upon realising that the functions should be reused in different cases, I started adding more parameters. The extra parameters added made the functions much more versatile as they could be reused for other situations by using different arguments. Decomposition of the problem, as taught during CZ1003, was extremely helpful since it gave each of us a sense of how one small function could be reused to solve different sub-tasks, and hence we knew what parameters to include inside a function.

5. Suggestions for Further Improvement

Improvements to the aesthetics of the application could increase user-friendliness and attractiveness to users. Also, the font size could be enlarged. Furthermore, more features such as the specialities of the stalls and available promotions could be implemented to encourage more users to use the application. Inclusion of a back button would make the application more convenient for users. Additionally, the previous windows can be closed automatically when another is opened.

6. Work distribution

	PIC
Main interface	Denise
Display available stalls	Denise
Set date, time	Anh
Menu at chosen date and time	Anh
Operating hours	Rachel
Estimated waiting	Rachel