Introduction to Computational Thinking

Mini Project

Real-time Canteen Information System

Tan Jun Hao U1923911J

Soh Qian Yi U1922306C

TABLE OF CONTENTS

1. Background

During peak hours, most canteens in NTU are especially crowded. Hence, a Real-time Informational NTU canteen system is proposed to solve this issue partially. Program allows user planned their time in advance and to make informed decision by showing important information about the store base on their input on so that they can stay productive throughout school term.

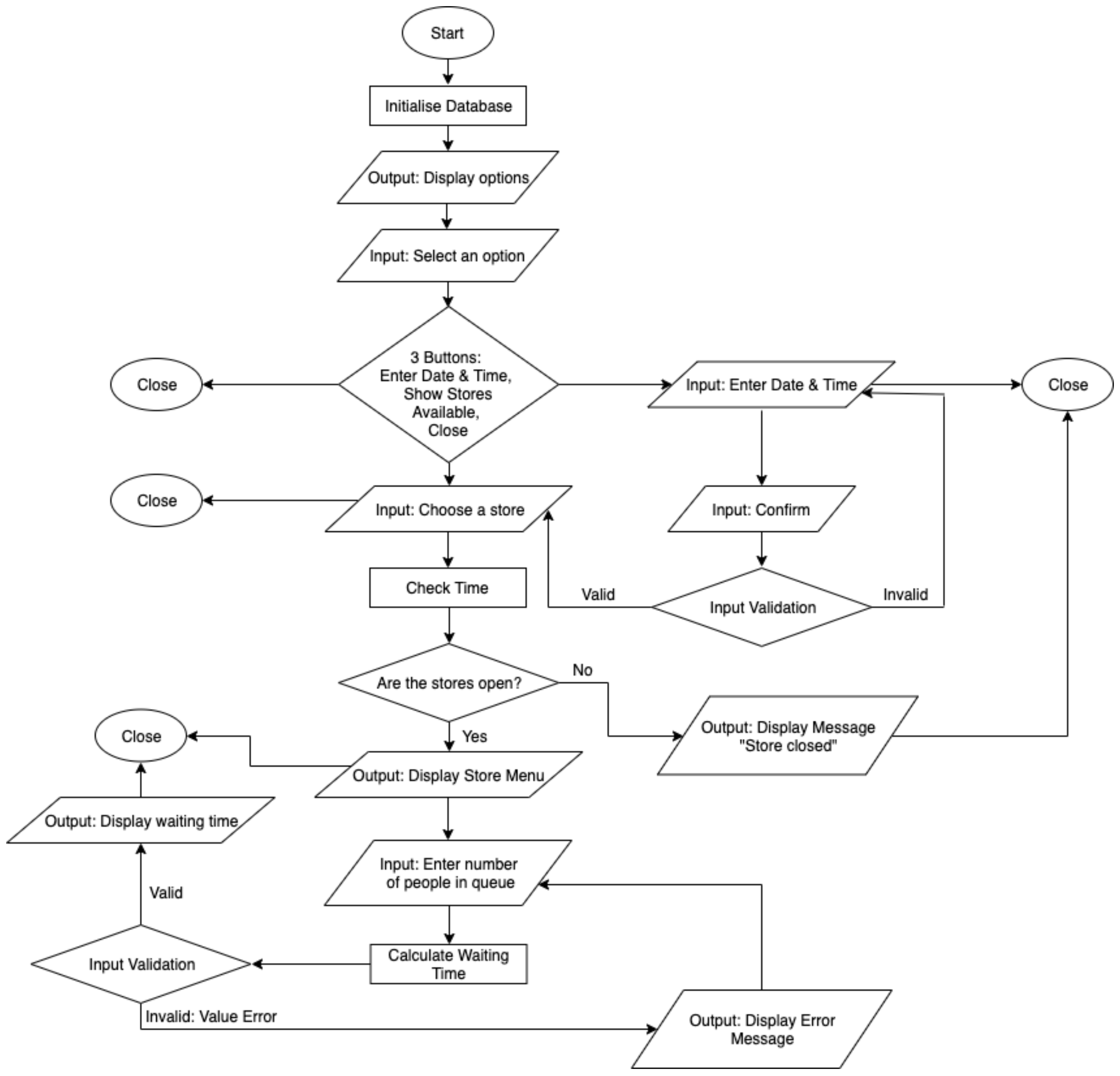
The system displays vital stall information and menu uses close time approximation accuracy (average time taken during the time and day of the week) to determine the waiting for a particular store in the canteen based on time of the day and day of a particular week. Moreover, system also allows variable change in the parameter of the store's existence, item being sold, its price, and days of operating hours.

2. Distribution of parts

Tan Jun Hao	Soh Qian Yi
Reading and parsing of details from database	Mainpage
Display all information & menu based on user-defined/system date & time	Database.txt
Waiting Time	User-defined date & time
Clock Set	Error handling
	User Interface

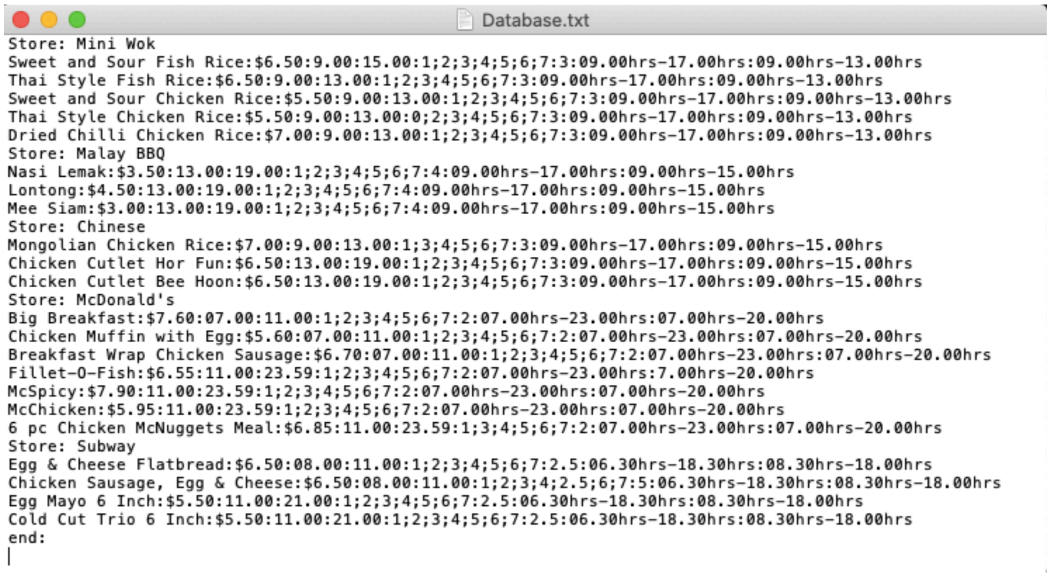
3. Algorithm Design

The following shows the top level flow chart of our system (**Fig. 1**).



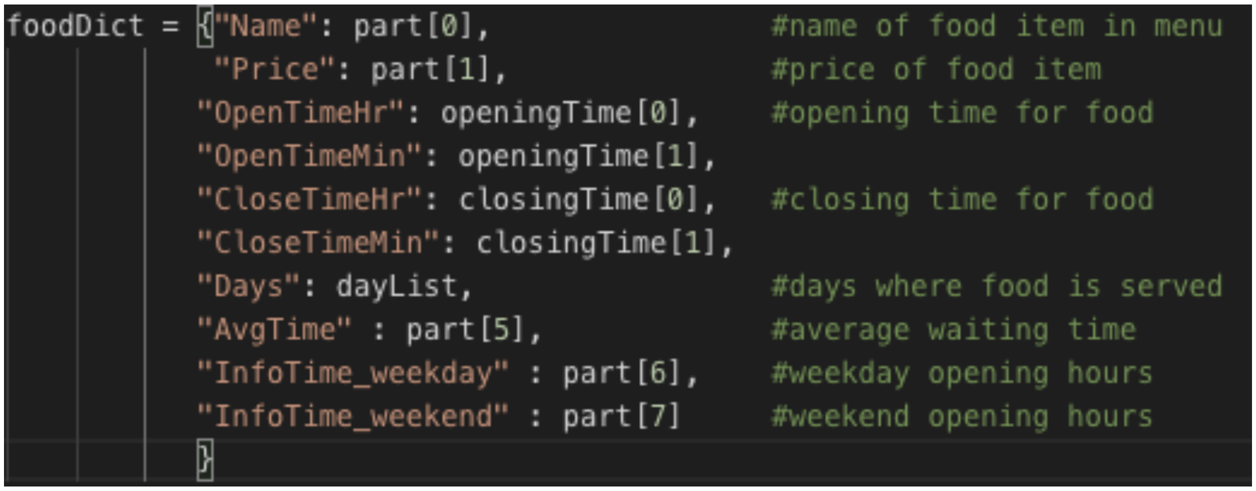
(Fig. 1)

This program is implemented using the Tkinter module. The information of each store, such as the menu and price for each food item was coded dynamically by using a .txt file (**Fig. 2**). Each part of the code is separated with a colon with the food name, price of food, starting timing of when the food will be served, ending timing of when the food will be served, days that the food will be served, average waiting time for food to be served, weekday opening hours and weekend opening hours respectively. A dictionary is also created as seen in **Fig. 2.1**.



```
Store: Mini Wok
Sweet and Sour Fish Rice:$6.50:9.00:15.00:1;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-13.00hrs
Thai Style Fish Rice:$6.50:9.00:13.00:1;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-13.00hrs
Sweet and Sour Chicken Rice:$5.50:9.00:13.00:1;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-13.00hrs
Thai Style Chicken Rice:$5.50:9.00:13.00:0;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-13.00hrs
Dried Chilli Chicken Rice:$7.00:9.00:13.00:1;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-13.00hrs
Store: Malay BBQ
Nasi Lemak:$3.50:13.00:19.00:1;2;3;4;5;6;7:4:09.00hrs-17.00hrs:09.00hrs-15.00hrs
Lontong:$4.50:13.00:19.00:1;2;3;4;5;6;7:4:09.00hrs-17.00hrs:09.00hrs-15.00hrs
Mee Siam:$3.00:13.00:19.00:1;2;3;4;5;6;7:4:09.00hrs-17.00hrs:09.00hrs-15.00hrs
Store: Chinese
Mongolian Chicken Rice:$7.00:9.00:13.00:1;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-15.00hrs
Chicken Cutlet Hor Fun:$6.50:13.00:19.00:1;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-15.00hrs
Chicken Cutlet Bee Hoon:$6.50:13.00:19.00:1;2;3;4;5;6;7:3:09.00hrs-17.00hrs:09.00hrs-15.00hrs
Store: McDonald's
Big Breakfast:$7.60:07.00:11.00:1;2;3;4;5;6;7:2:07.00hrs-23.00hrs:07.00hrs-20.00hrs
Chicken Muffin with Egg:$5.60:07.00:11.00:1;2;3;4;5;6;7:2:07.00hrs-23.00hrs:07.00hrs-20.00hrs
Breakfast Wrap Chicken Sausage:$6.70:07.00:11.00:1;2;3;4;5;6;7:2:07.00hrs-23.00hrs:07.00hrs-20.00hrs
Fillet-O-Fish:$6.55:11.00:23.59:1;2;3;4;5;6;7:2:07.00hrs-23.00hrs:7.00hrs-20.00hrs
McSpicy:$7.90:11.00:23.59:1;2;3;4;5;6;7:2:07.00hrs-23.00hrs:07.00hrs-20.00hrs
McChicken:$5.95:11.00:23.59:1;2;3;4;5;6;7:2:07.00hrs-23.00hrs:07.00hrs-20.00hrs
6 pc Chicken McNuggets Meal:$6.85:11.00:23.59:1;3;4;5;6;7:2:07.00hrs-23.00hrs:07.00hrs-20.00hrs
Store: Subway
Egg & Cheese Flatbread:$6.50:08.00:11.00:1;2;3;4;5;6;7:2.5:06.30hrs-18.30hrs:08.30hrs-18.00hrs
Chicken Sausage, Egg & Cheese:$6.50:08.00:11.00:1;2;3;4;2.5;6;7:5:06.30hrs-18.30hrs:08.30hrs-18.00hrs
Egg Mayo 6 Inch:$5.50:11.00:21.00:1;2;3;4;5;6;7:2.5:06.30hrs-18.30hrs:08.30hrs-18.00hrs
Cold Cut Trio 6 Inch:$5.50:11.00:21.00:1;2;3;4;5;6;7:2.5:06.30hrs-18.30hrs:08.30hrs-18.00hrs
end:
```

(Fig. 2)



```
foodDict = {"Name": part[0],                      #name of food item in menu
             "Price": part[1],                    #price of food item
            "OpenTimeHr": openingTime[0],         #opening time for food
            "OpenTimeMin": openingTime[1],
            "CloseTimeHr": closingTime[0],        #closing time for food
            "CloseTimeMin": closingTime[1],
            "Days": dayList,                      #days where food is served
            "AvgTime" : part[5],                  #average waiting time
            "InfoTime_weekday" : part[6],         #weekday opening hours
            "InfoTime_weekend" : part[7]          #weekend opening hours
            }
```

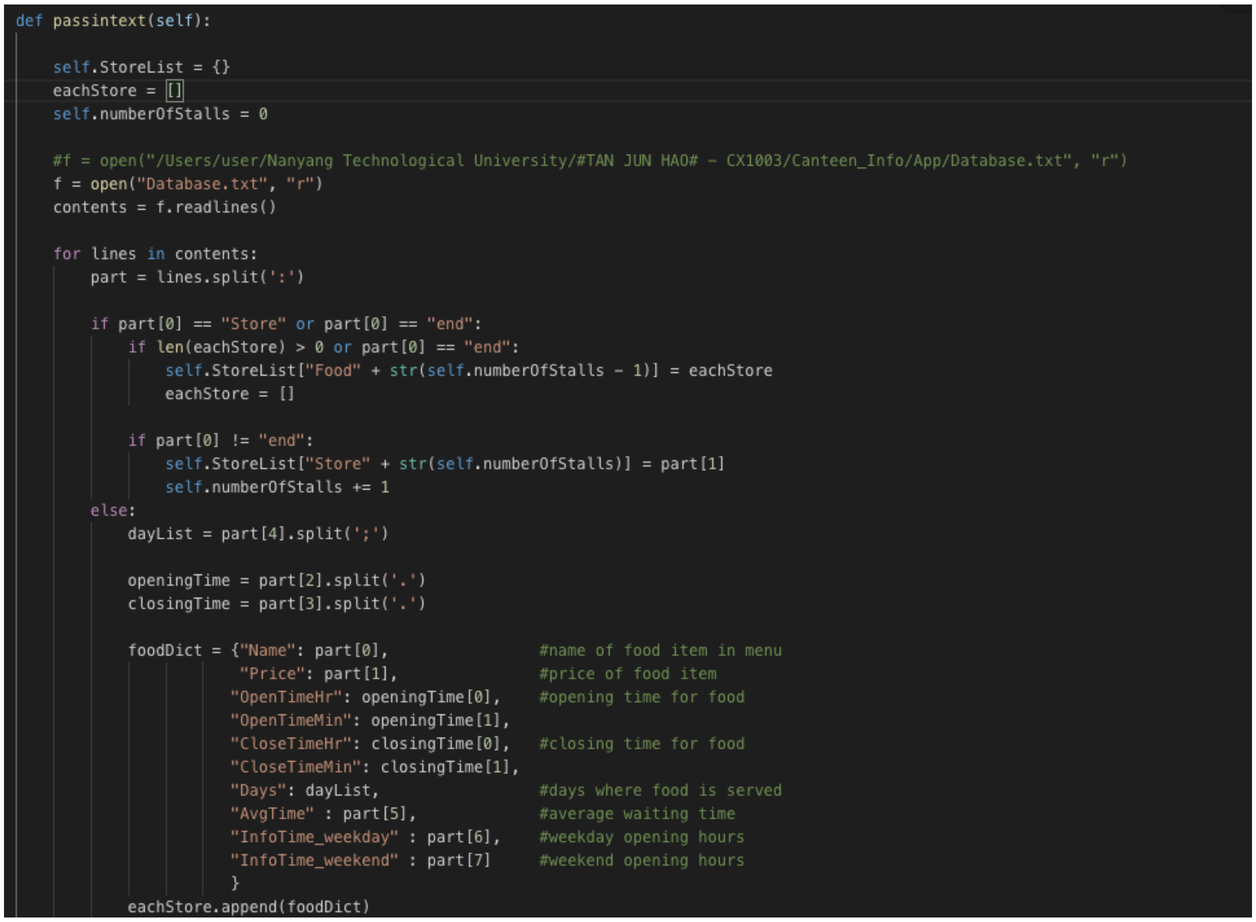
(Fig. 2.1)

4. Brief description of user-defined functions

4.1 Passintext (Fig. 3)

User defined function “passintext” integrates for-loop with nested if statements for file text reading iteration. Data from database file is parse through the passintext function. Values are then progressively saved into the list as dictionary key which can then be called repeated and to be used in the program. This forms the backbone of this program structure.

If the first index of the Database.txt file is “Store” and if length of list eachStore >0, then the program will continue storing each lines of iteration down. Furthering in each line of storage, every parameter is being segregated by “:” for easy reference calling. Once the program reaches the “end” it will stop iterating through the lines of code. All stores are stored in StoreList{} while all parameters are stored in eachStore[].

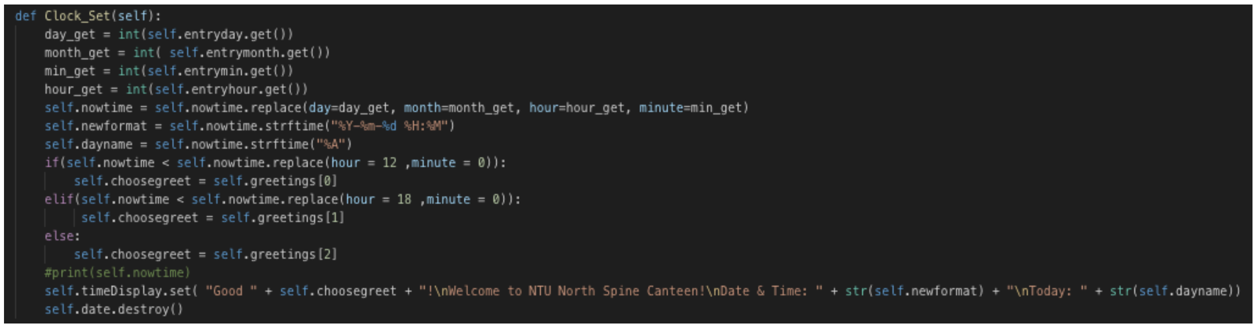


```python
def passintext(self):

    self.StoreList = {}
    eachStore = []
    self.numberOfStalls = 0

    #f = open("/Users/user/Nanyang Technological University/#TAN JUN HAO# - CX1003/Canteen_Info/App/Database.txt", "r")
    f = open("Database.txt", "r")
    contents = f.readlines()

    for lines in contents:
        part = lines.split(':')

        if part[0] == "Store" or part[0] == "end":
            if len(eachStore) > 0 or part[0] == "end":
                self.StoreList["Food" + str(self.numberOfStalls - 1)] = eachStore
                eachStore = []

            if part[0] != "end":
                self.StoreList["Store" + str(self.numberOfStalls)] = part[1]
                self.numberOfStalls += 1
        else:
            dayList = part[4].split(';')

            openingTime = part[2].split('.')
            closingTime = part[3].split('.')

            foodDict = {"Name": part[0],                #name of food item in menu
                         "Price": part[1],              #price of food item
                        "OpenTimeHr": openingTime[0],   #opening time for food
                        "OpenTimeMin": openingTime[1],
                        "CloseTimeHr": closingTime[0],  #closing time for food
                        "CloseTimeMin": closingTime[1],
                        "Days": dayList,                #days where food is served
                        "AvgTime" : part[5],            #average waiting time
                        "InfoTime_weekday" : part[6],   #weekday opening hours
                        "InfoTime_weekend" : part[7]    #weekend opening hours
                        }
            eachStore.append(foodDict)
```

(Fig. 3)

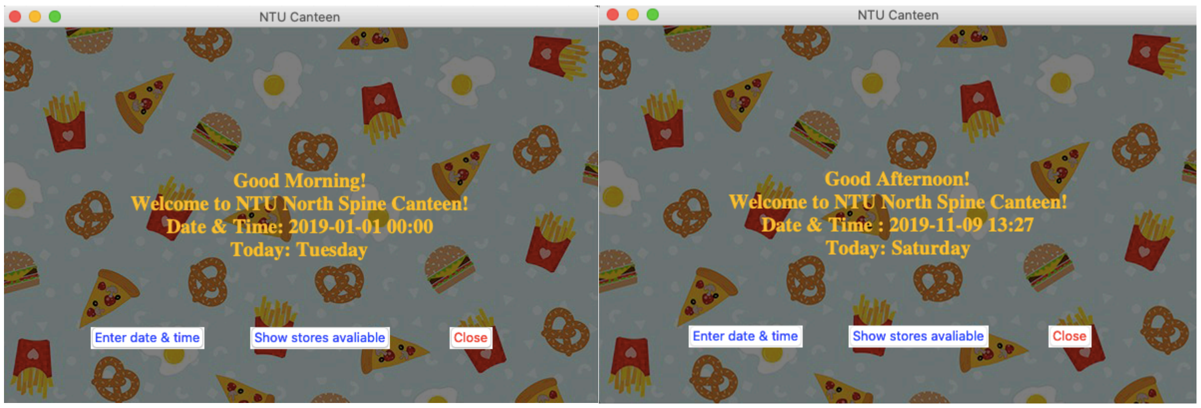
4.2 Clock_Set (Fig. 4)

This function retrieves the data from user defined date and time which replaces the system date and time. If the user inputs a timing before 1200 Hours, 1800 Hours and 2359 Hours, the program would return a page with a morning, afternoon and evening greeting respectively. This would overwrite the system date and time and display stall information and menu based on user defined date and time accordingly.



```python
def Clock_Set(self):
    day_get = int(self.entryday.get())
    month_get = int( self.entrymonth.get())
    min_get = int(self.entrymin.get())
    hour_get = int(self.entryhour.get())
    self.nowtime = self.nowtime.replace(day=day_get, month=month_get, hour=hour_get, minute=min_get)
    self.newformat = self.nowtime.strftime("%Y-%m-%d %H:%M")
    self.dayname = self.nowtime.strftime("%A")
    if(self.nowtime < self.nowtime.replace(hour = 12 ,minute = 0)):
        self.choosegreet = self.greetings[0]
    elif(self.nowtime < self.nowtime.replace(hour = 18 ,minute = 0)):
         self.choosegreet = self.greetings[1]
    else:
        self.choosegreet = self.greetings[2]
    #print(self.nowtime)
    self.timeDisplay.set( "Good " + self.choosegreet + "!\nWelcome to NTU North Spine Canteen!\nDate & Time: " + str(self.newformat) + "\nToday: " + str(self.dayname))
    self.date.destroy()
```

(Fig. 4)



```
Good Morning!
Welcome to NTU North Spine Canteen!
Date & Time: 2019-01-01 00:00
Today: Tuesday
```

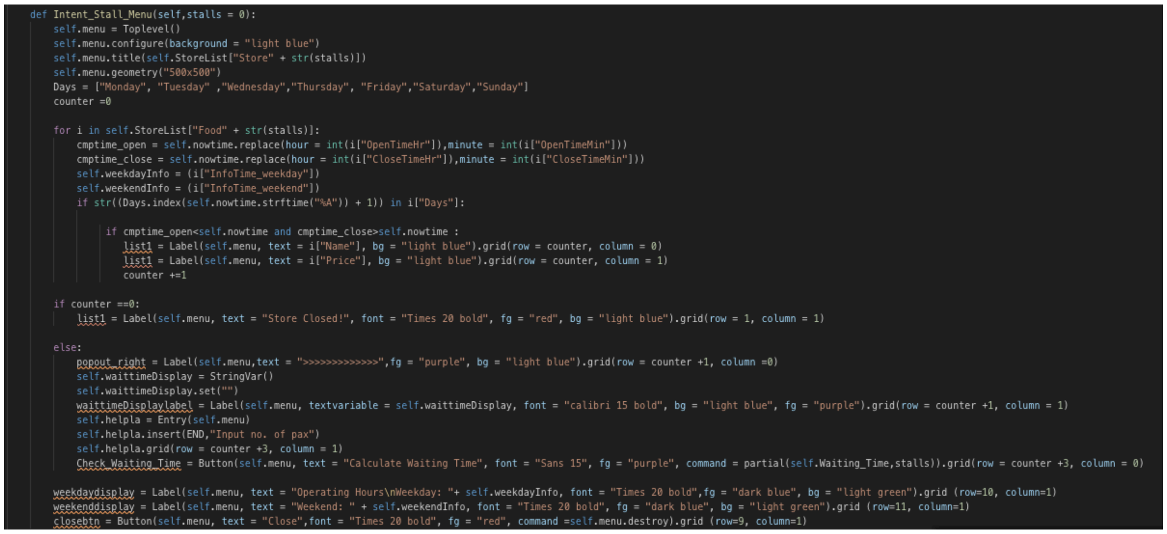
```
Good Afternoon!
Welcome to NTU North Spine Canteen!
Date & Time : 2019-11-09 13:27
Today: Saturday
```

(Fig 4.1: Example of output)

4.3 Intent_Stall_Menu (Fig. 5)

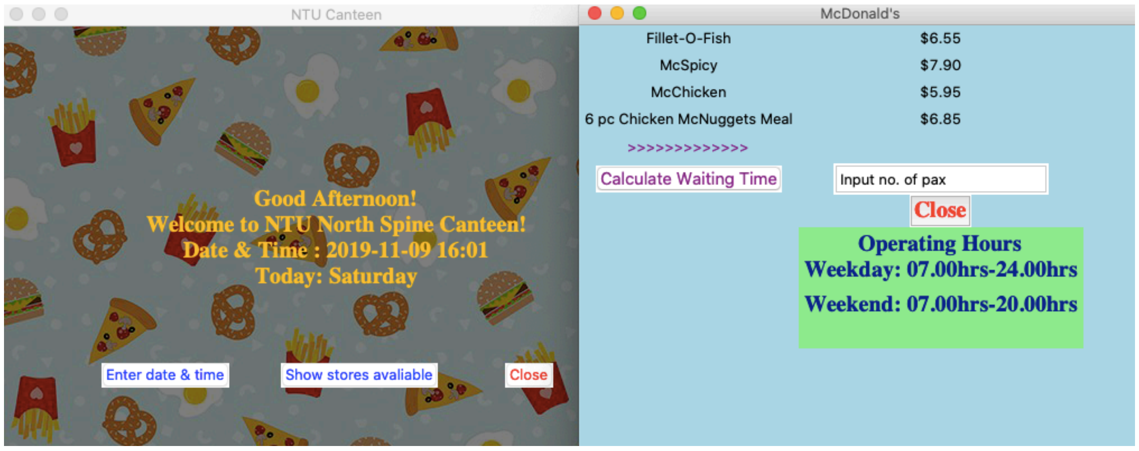
This function compares the system timing or user-defined timing with the opening hour and closing hours of the food item served, which will then be shown as the food menu if both conditions are met. (i.e if system time/user-defined time is within the opening and closing hours of the food) Counter will be increased by 1.

If counter = 0, it would mean that the store is closed as system timing/user-defined timing is not within the opening hours of the store. Otherwise if store is not closed, user will be able to input number of people in the queue to calculate the estimated waiting time.



```python
def Intent_Stall_Menu(self,stalls = 0):
    self.menu = Toplevel()
    self.menu.configure(background = "light blue")
    self.menu.title(self.StoreList["Store" + str(stalls)])
    self.menu.geometry("500x500")
    Days = ["Monday", "Tuesday" ,"Wednesday","Thursday", "Friday","Saturday","Sunday"]
    counter =0

    for i in self.StoreList["Food" + str(stalls)]:
        cmptime_open = self.nowtime.replace(hour = int(i["OpenTimeHr"]),minute = int(i["OpenTimeMin"]))
        cmptime_close = self.nowtime.replace(hour = int(i["CloseTimeHr"]),minute = int(i["CloseTimeMin"]))
        self.weekdayInfo = (i["InfoTime_weekday"])
        self.weekendInfo = (i["InfoTime_weekend"])
        if str((Days.index(self.nowtime.strftime("%A")) + 1)) in i["Days"]:

            if cmptime_open<self.nowtime and cmptime_close>self.nowtime :
                list1 = Label(self.menu, text = i["Name"], bg = "light blue").grid(row = counter, column = 0)
                list1 = Label(self.menu, text = i["Price"], bg = "light blue").grid(row = counter, column = 1)
                counter +=1

    if counter ==0:
        list1 = Label(self.menu, text = "Store Closed!", font = "Times 20 bold", fg = "red", bg = "light blue").grid(row = 1, column = 1)

    else:
        popout_right = Label(self.menu,text = ">>>>>>>>>>>>>",fg = "purple", bg = "light blue").grid(row = counter +1, column =0)
        self.waittimeDisplay = StringVar()
        self.waittimeDisplay.set("")
        waittimeDisplaylabel = Label(self.menu, textvariable = self.waittimeDisplay, font = "calibri 15 bold", bg = "light blue", fg = "purple").grid(row = counter +1, column = 1)
        self.helpla = Entry(self.menu)
        self.helpla.insert(END,"Input no. of pax")
        self.helpla.grid(row = counter +3, column = 1)
        Check_Waiting_Time = Button(self.menu, text = "Calculate Waiting Time", font = "Sans 15", fg = "purple", command = partial(self.Waiting_Time,stalls)).grid(row = counter +3, column = 0)

    weekdaydisplay = Label(self.menu, text = "Operating Hours\nWeekday: "+ self.weekdayInfo, font = "Times 20 bold",fg = "dark blue", bg = "light green").grid (row=10, column=1)
    weekenddisplay = Label(self.menu, text = "Weekend: " + self.weekendInfo, font = "Times 20 bold", fg = "dark blue", bg = "light green").grid (row=11, column=1)
    closebtn = Button(self.menu, text = "Close",font = "Times 20 bold", fg = "red", command =self.menu.destroy).grid (row=9, column=1)
```

(Fig. 5)



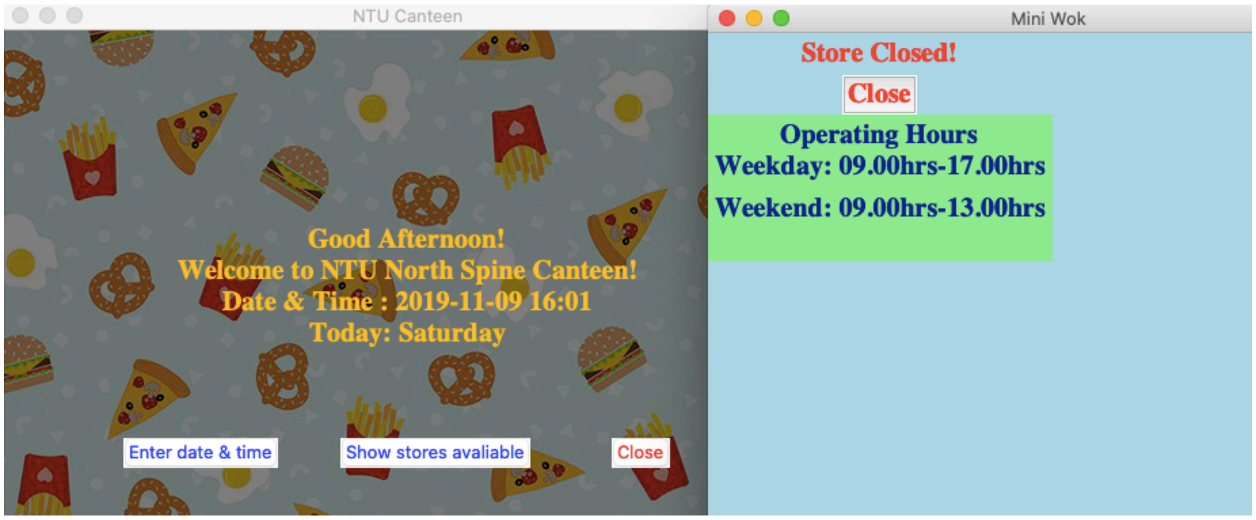
Good Afternoon!
Welcome to NTU North Spine Canteen!
Date & Time : 2019-11-09 16:01
Today: Saturday

Fillet-O-Fish	$6.55
McSpicy	$7.90
McChicken	$5.95
6 pc Chicken McNuggets Meal	$6.85

>>>>>>>>>>>>>

Operating Hours
Weekday: 07.00hrs-24.00hrs
Weekend: 07.00hrs-20.00hrs

(Fig. 5.1: Example of output when store is open)



Good Afternoon!
Welcome to NTU North Spine Canteen!
Date & Time : 2019-11-09 16:01
Today: Saturday

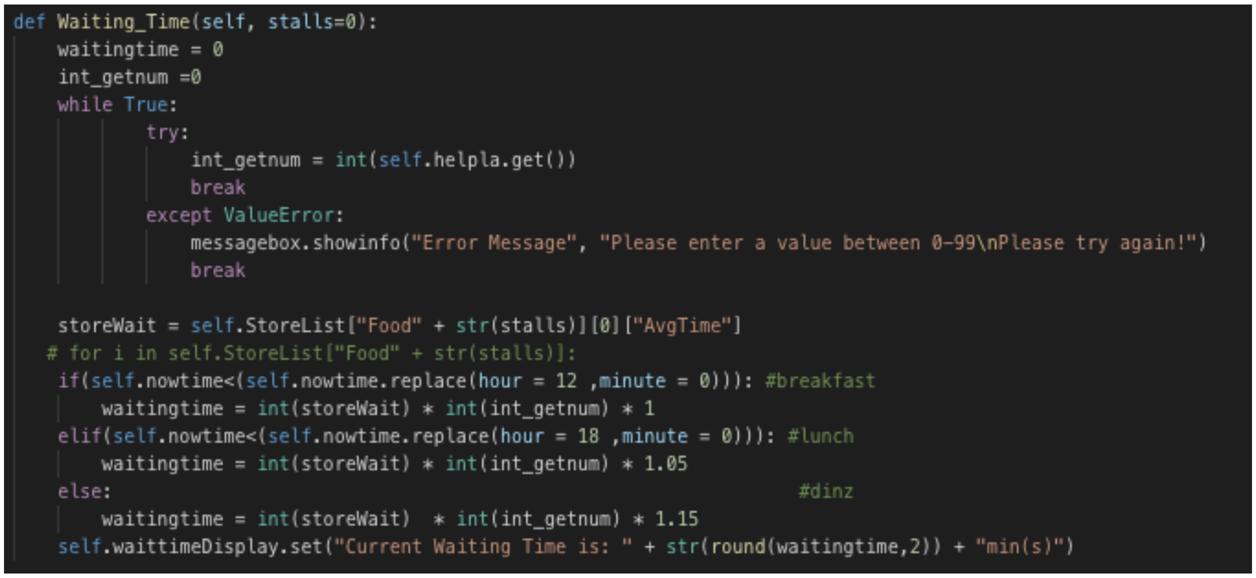
Store Closed!

Operating Hours
Weekday: 09.00hrs-17.00hrs
Weekend: 09.00hrs-13.00hrs

(Fig. 5.2: Example of output when store is closed)

4.4 Waiting_Time (Fig. 6)

This function includes the algorithm for waiting the estimated waiting time based on the user's input on the number of people in the queue. For each store, the average waiting time differs. Hence, the waiting time will be calculated accordingly using the average waiting time for each store and depends on the timing they order. For example, waiting time for users to get their food would be longer during dinner time as compared to breakfast and lunch timing. Program testing will be discussed in the next section.

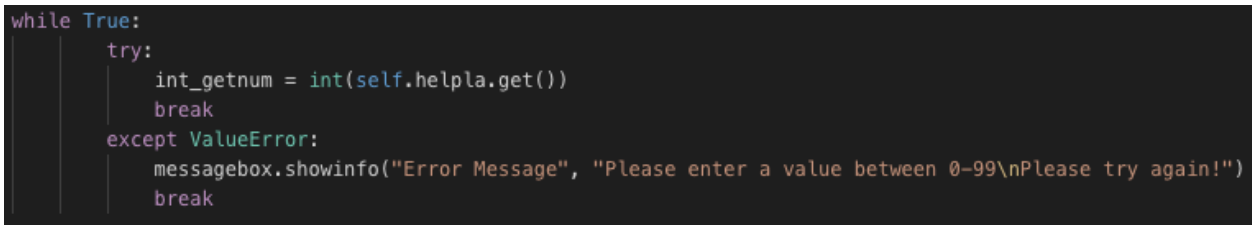


```
def Waiting_Time(self, stalls=0):
    waitingtime = 0
    int_getnum =0
    while True:
            try:
                int_getnum = int(self.helpla.get())
                break
            except ValueError:
                messagebox.showinfo("Error Message", "Please enter a value between 0-99\nPlease try again!")
                break

    storeWait = self.StoreList["Food" + str(stalls)][0]["AvgTime"]
   # for i in self.StoreList["Food" + str(stalls)]:
    if(self.nowtime<(self.nowtime.replace(hour = 12 ,minute = 0))): #breakfast
        waitingtime = int(storeWait) * int(int_getnum) * 1
    elif(self.nowtime<(self.nowtime.replace(hour = 18 ,minute = 0))): #lunch
        waitingtime = int(storeWait) * int(int_getnum) * 1.05
    else:                                                          #dinz
        waitingtime = int(storeWait)  * int(int_getnum) * 1.15
    self.waittimeDisplay.set("Current Waiting Time is: " + str(round(waitingtime,2)) + "min(s)")
```

(Fig. 6)

5. Program testing

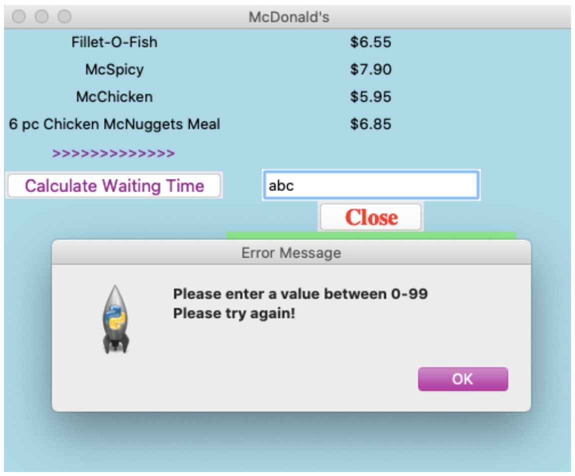
By using exception handling process, we have used the try-except method to handle errors by the user. The figure below shows our code (**Fig. 7.1**).



```
while True:
    try:
        int_getnum = int(self.helpla.get())
        break
    except ValueError:
        messagebox.showinfo("Error Message", "Please enter a value between 0-99\nPlease try again!")
        break
```

(Fig. 7.1)

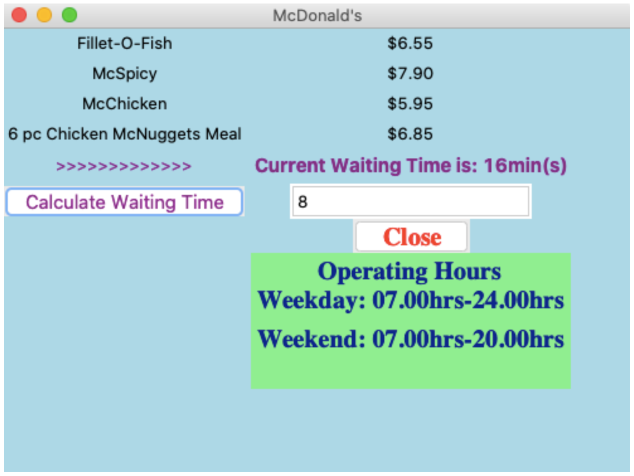
When the user inputs another value instead of an integer, a message box will appear, prompting the user to only input integers (**Fig. 7.2**). However, if the user correctly enters an integer, the system will proceed to calculate the average waiting time and display it to the user (**Fig. 7.3**).



McDonald's

Fillet-O-Fish	$6.55
McSpicy	$7.90
McChicken	$5.95
6 pc Chicken McNuggets Meal	$6.85

Please enter a value between 0-99
Please try again!



McDonald's

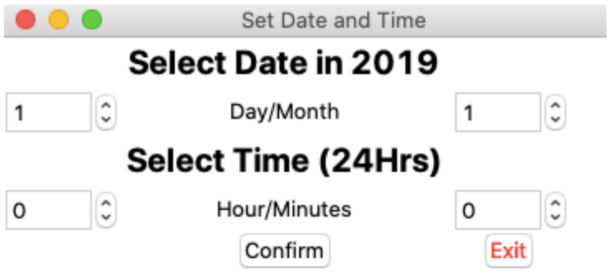
Fillet-O-Fish	$6.55
McSpicy	$7.90
McChicken	$5.95
6 pc Chicken McNuggets Meal	$6.85

Current Waiting Time is: 16min(s)

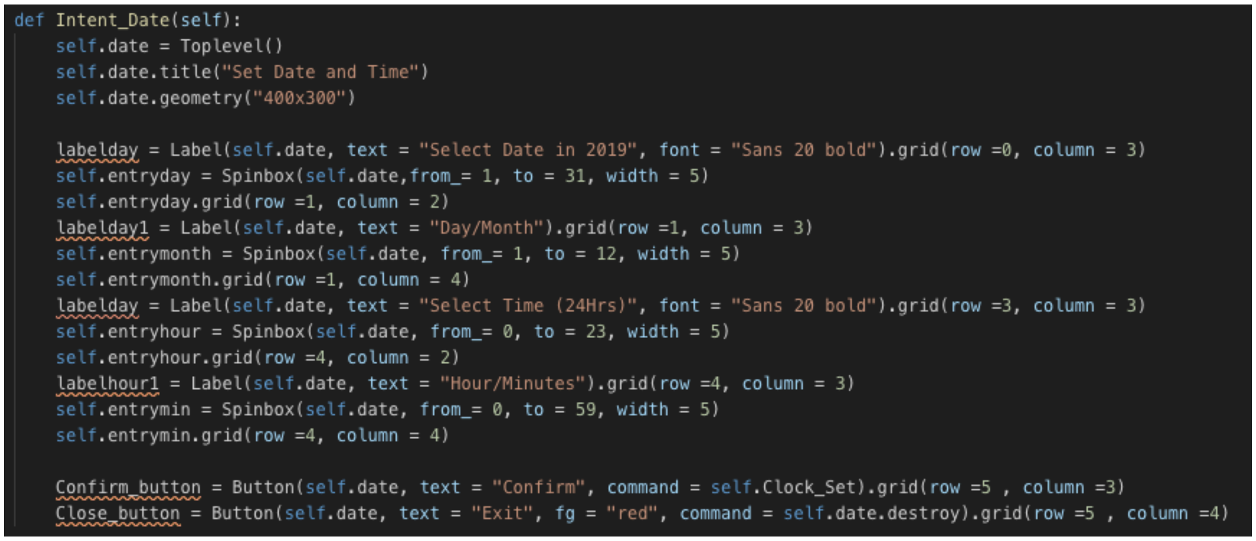
Operating Hours
Weekday: 07.00hrs-24.00hrs
Weekend: 07.00hrs-20.00hrs

(Fig. 7.2) **(Fig. 7.3)**

Meanwhile, for the user defined input time and date, we have opted to use a spin box widget instead. Hence, minimizing the possibility of the user keying in wrong data (**Fig. 8**). **Fig. 8.1** shows the code that we have used for the spin box widget.



(Fig. 8)



```
def Intent_Date(self):
    self.date = Toplevel()
    self.date.title("Set Date and Time")
    self.date.geometry("400x300")

    labelday = Label(self.date, text = "Select Date in 2019", font = "Sans 20 bold").grid(row =0, column = 3)
    self.entryday = Spinbox(self.date,from_= 1, to = 31, width = 5)
    self.entryday.grid(row =1, column = 2)
    labelday1 = Label(self.date, text = "Day/Month").grid(row =1, column = 3)
    self.entrymonth = Spinbox(self.date, from_= 1, to = 12, width = 5)
    self.entrymonth.grid(row =1, column = 4)
    labelday = Label(self.date, text = "Select Time (24Hrs)", font = "Sans 20 bold").grid(row =3, column = 3)
    self.entryhour = Spinbox(self.date, from_= 0, to = 23, width = 5)
    self.entryhour.grid(row =4, column = 2)
    labelhour1 = Label(self.date, text = "Hour/Minutes").grid(row =4, column = 3)
    self.entrymin = Spinbox(self.date, from_= 0, to = 59, width = 5)
    self.entrymin.grid(row =4, column = 4)

    Confirm_button = Button(self.date, text = "Confirm", command = self.Clock_Set).grid(row =5 , column =3)
    Close_button = Button(self.date, text = "Exit", fg = "red", command = self.date.destroy).grid(row =5 , column =4)
```

(Fig. 8.1)

6. Reflections

The primary focus of programming this python application is to be as dynamic as possible with futureproofing in mind. This would allow the user to update any argument as per required when condition changes. Changing conditions includes but are not limited to the store names, item, items prices, …etc. With code structure dynamism involved, the codes are persistently reused which resulted in less code output and therefore, less susceptible to error which, achieved the desired results with every condition met. Overall, this allows for debugging to be easier as lesser code is involved and worry less about its dependencies – which would be tedious to find the root case of the issue when codes are interlinked with each other for the program to work as expected.

As per project requirements, integrating all the modules within Tkinter is a slight hurdle as knowledge from both Python and Tkinter is required. While Python, like many other programming language, has a similar structure, it's syntax is confusing as it oversimplifies function instantiation which causes hick ups in the beginning stages. Coupled with Tkinter integration, references must be made simultaneously while developing the program.

In conclusion, we started off with many doubts and uncertainty on this project. The many difficulties that we have faced since the start has allowed us to learn python in greater scope. By researching online tutorials and seeking some guidance from our peers, we have managed to overcome every issue, hurdle by hurdle. Finally, we managed to design and code a working program from scratch with every requirement met as stated in the project scope.