NANYANG TECHNOLOGICAL UNIVERSITY

Title:

Real-Time Canteen Information System

NTU Foodies Application

By:

Tutorial Group: FS5

Team members (Student ID):

Lim Bin Hong Jasper (U1922783B)

Li Hao Cheng (U1921700J)

Khin Nway Htway (U1921268F)

In partial fulfilment of the requirements for the module

CZ1003: Introduction to Computational Thinking

12 November 2019

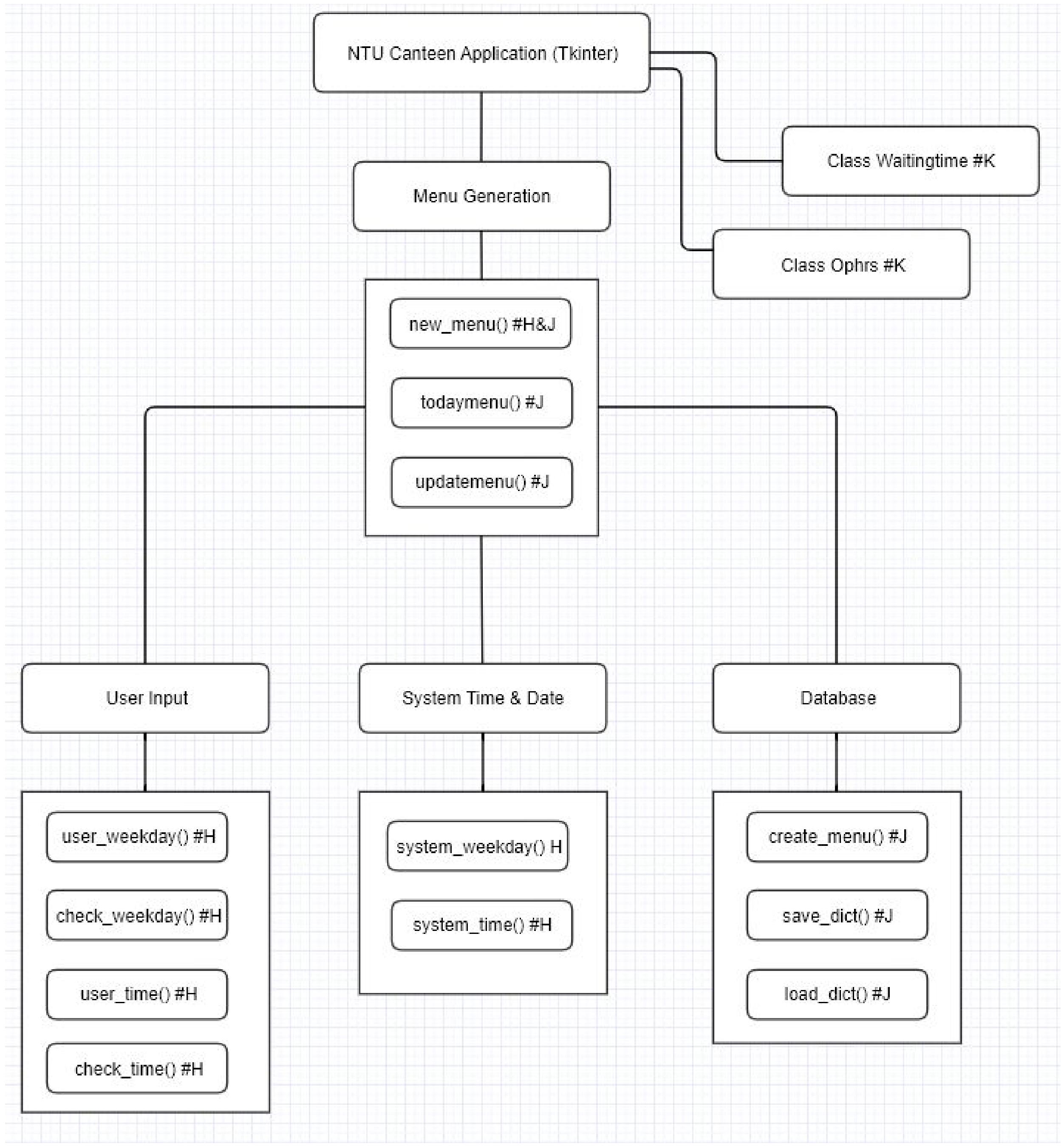
Content page

List of Figures

Introduction

This report aims to provide an informative report on the ‘NTU Foodies’ application created in python with Tkinter as the main GUI library. It starts by showing a top-down module implementation map and how the task is distributed. Next we will go into the individual functions that make up the application, followed by test cases that were done to check the error handling of the program. Finally the reflection of each of the individual members will conclude this report.

Top-down Module Map

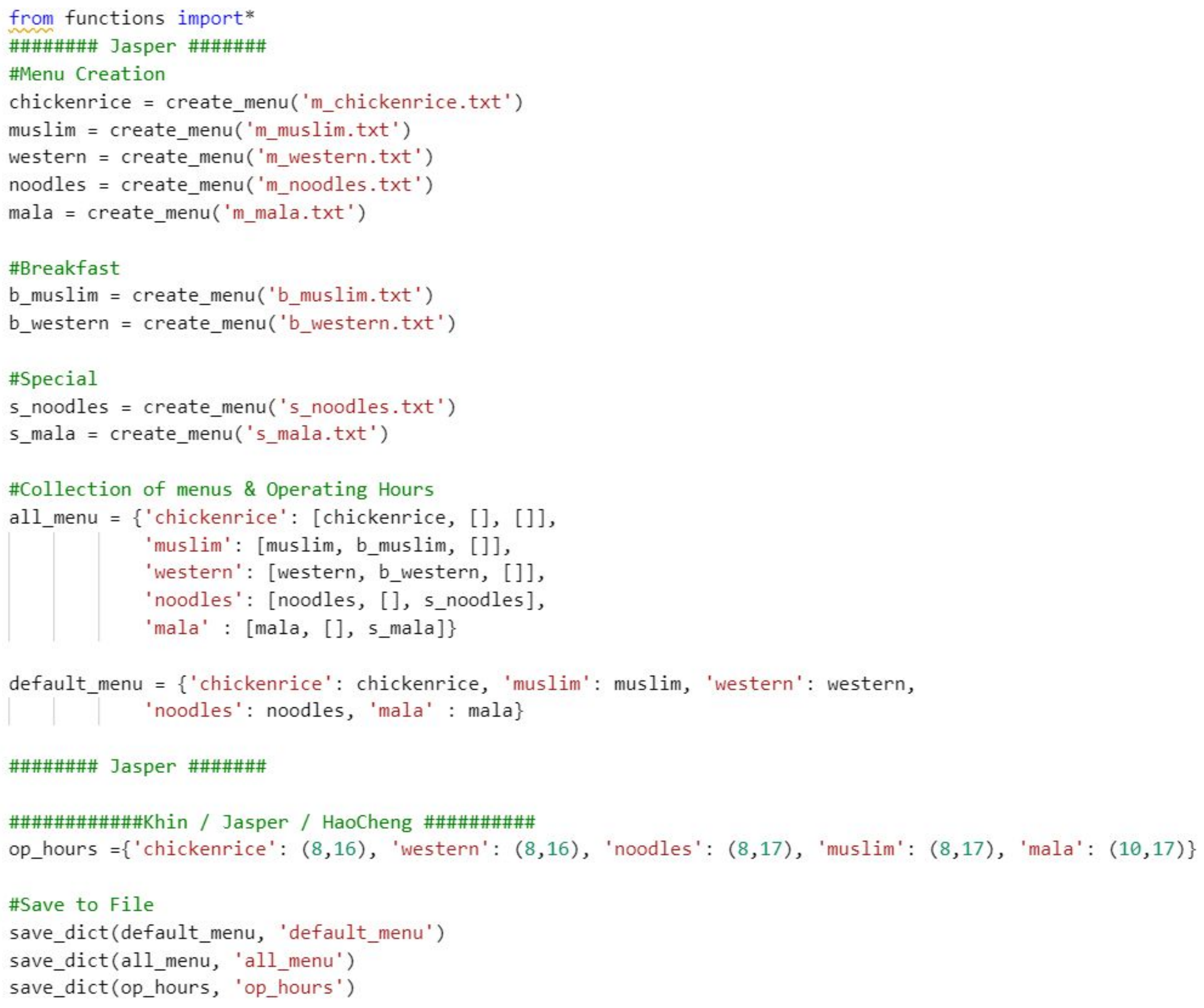


Legend : K = khin, H = haocheng, J = Jasper

Figure 1: Top-down Module Map

Data

The program focuses on updating default_menu with menus listed in all_menu.



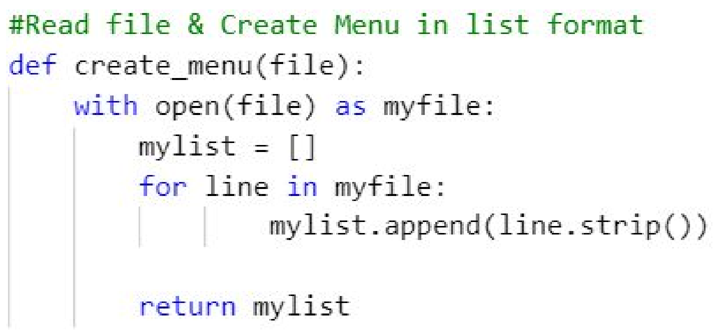
```python
from functions import*
######## Jasper #######
#Menu Creation
chickenrice = create_menu('m_chickenrice.txt')
muslim = create_menu('m_muslim.txt')
western = create_menu('m_western.txt')
noodles = create_menu('m_noodles.txt')
mala = create_menu('m_mala.txt')

#Breakfast
b_muslim = create_menu('b_muslim.txt')
b_western = create_menu('b_western.txt')

#Special
s_noodles = create_menu('s_noodles.txt')
s_mala = create_menu('s_mala.txt')

#Collection of menus & Operating Hours
all_menu = {'chickenrice': [chickenrice, [], []],
            'muslim': [muslim, b_muslim, []],
            'western': [western, b_western, []],
            'noodles': [noodles, [], s_noodles],
            'mala' : [mala, [], s_mala]}

default_menu = {'chickenrice': chickenrice, 'muslim': muslim, 'western': western,
            'noodles': noodles, 'mala' : mala}

######## Jasper #######

############Khin / Jasper / HaoCheng ##########
op_hours ={'chickenrice': (8,16), 'western': (8,16), 'noodles': (8,17), 'muslim': (8,17), 'mala': (10,17)}

#Save to File
save_dict(default_menu, 'default_menu')
save_dict(all_menu, 'all_menu')
save_dict(op_hours, 'op_hours')
```

Figure 2: database.py

Modules

#Jasper#

1. Function: save_dict(data,file)

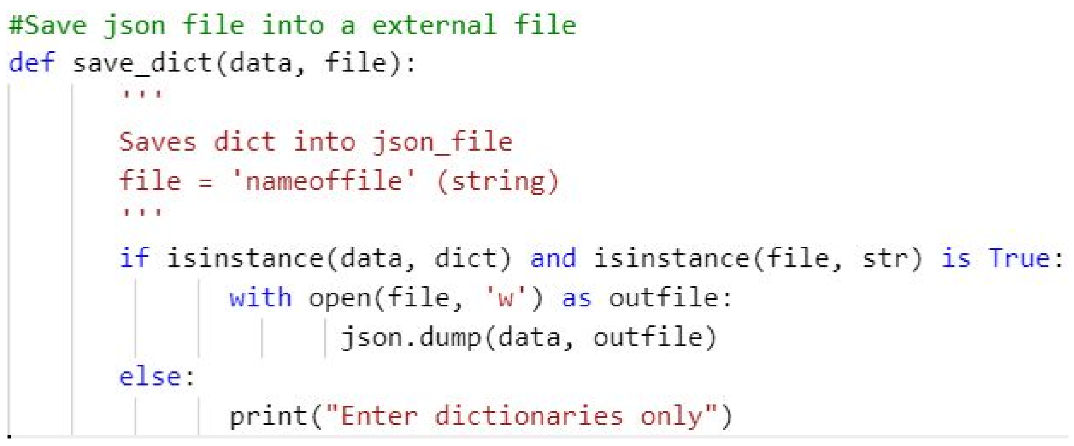


```python
#Read file & Create Menu in list format
def create_menu(file):
    with open(file) as myfile:
        mylist = []
        for line in myfile:
            mylist.append(line.strip())

        return mylist
```

Figure 3: save_dict()

The idea behind this function was to allow future changes to the stalls menu without the need to edit the main program code, it generates a list of items in the text file.

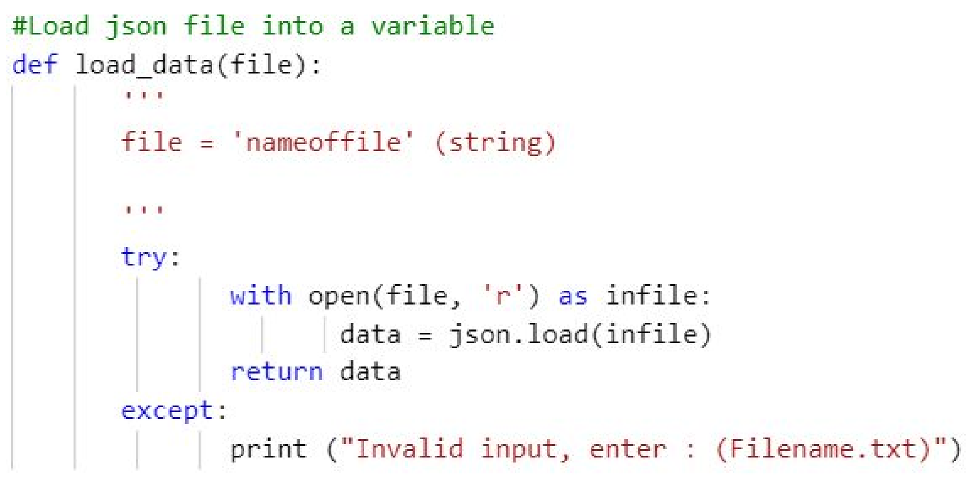
2. save_dict(data,file)



```python
#Save json file into a external file
def save_dict(data, file):
        '''
        Saves dict into json_file
        file = 'nameoffile' (string)
        '''
        if isinstance(data, dict) and isinstance(file, str) is True:
            with open(file, 'w') as outfile:
                json.dump(data, outfile)
        else:
            print("Enter dictionaries only")
```

Figure 4: save_dict() function

3. load_data(file)

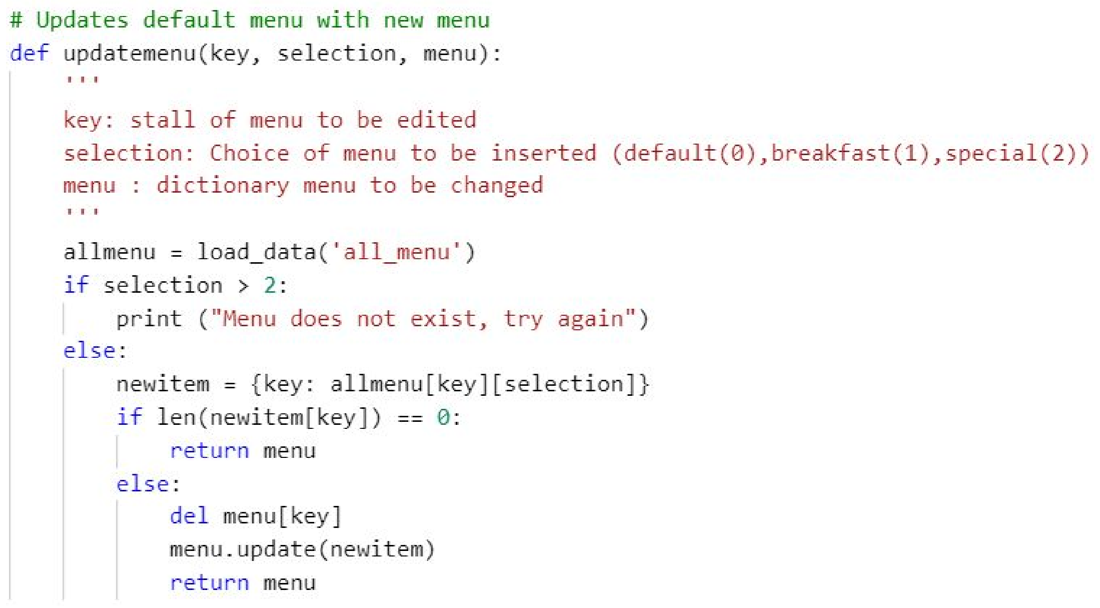


```
#Load json file into a variable
def load_data(file):
        '''
        file = 'nameoffile' (string)

        '''
        try:
                with open(file, 'r') as infile:
                        data = json.load(infile)
                return data
        except:
                print ("Invalid input, enter : (Filename.txt)")
```

Figure 5: load_data() function

The two functions save_dict() and load_data() work in conjunction with each other to save and load our main database file(Json format). This would allow members to use the database by loading the core files.

4. updatemenu(key,selection,default = load_data ('default menu'))

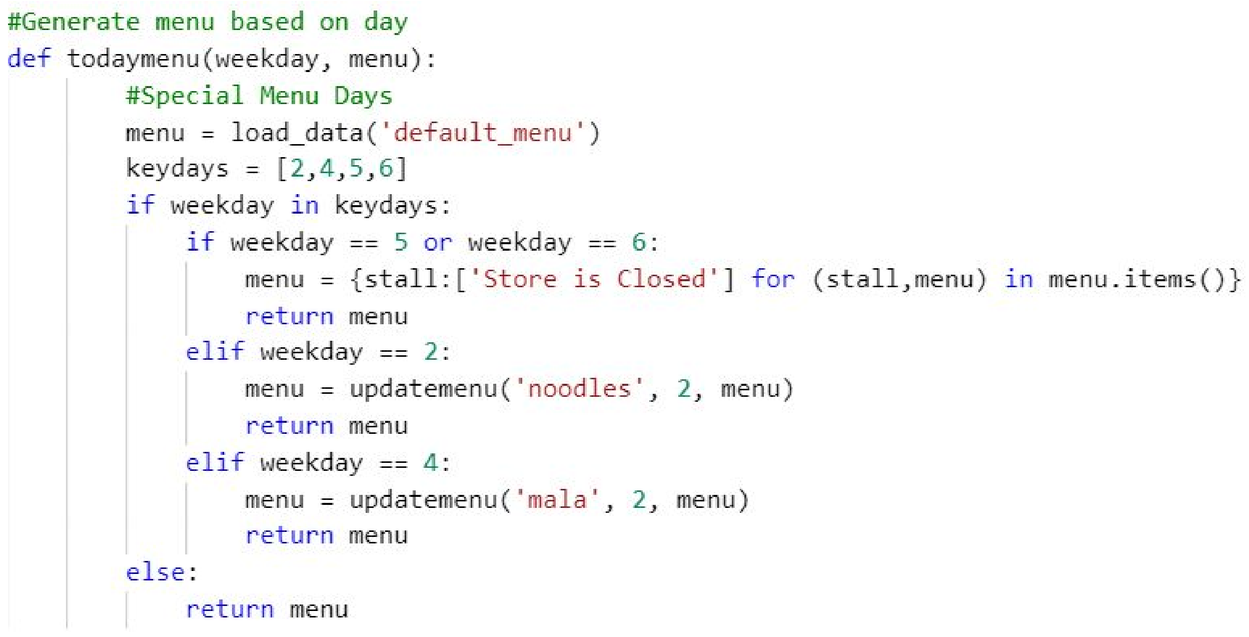


```
# Updates default menu with new menu
def updatemenu(key, selection, menu):
    '''
    key: stall of menu to be edited
    selection: Choice of menu to be inserted (default(0),breakfast(1),special(2))
    menu : dictionary menu to be changed
    '''
    allmenu = load_data('all_menu')
    if selection > 2:
        print ("Menu does not exist, try again")
    else:
        newitem = {key: allmenu[key][selection]}
        if len(newitem[key]) == 0:
            return menu
        else:
            del menu[key]
            menu.update(newitem)
            return menu
```

Figure 6: updatemenu() function

This function generates a key:value pairs taken from all_menu and replaces the current one in the menu passed in. It is used to update the menu based on conditions given, and checks if a valid selection if entered, if not a default menu is returned.

5. todaymenu(weekday)



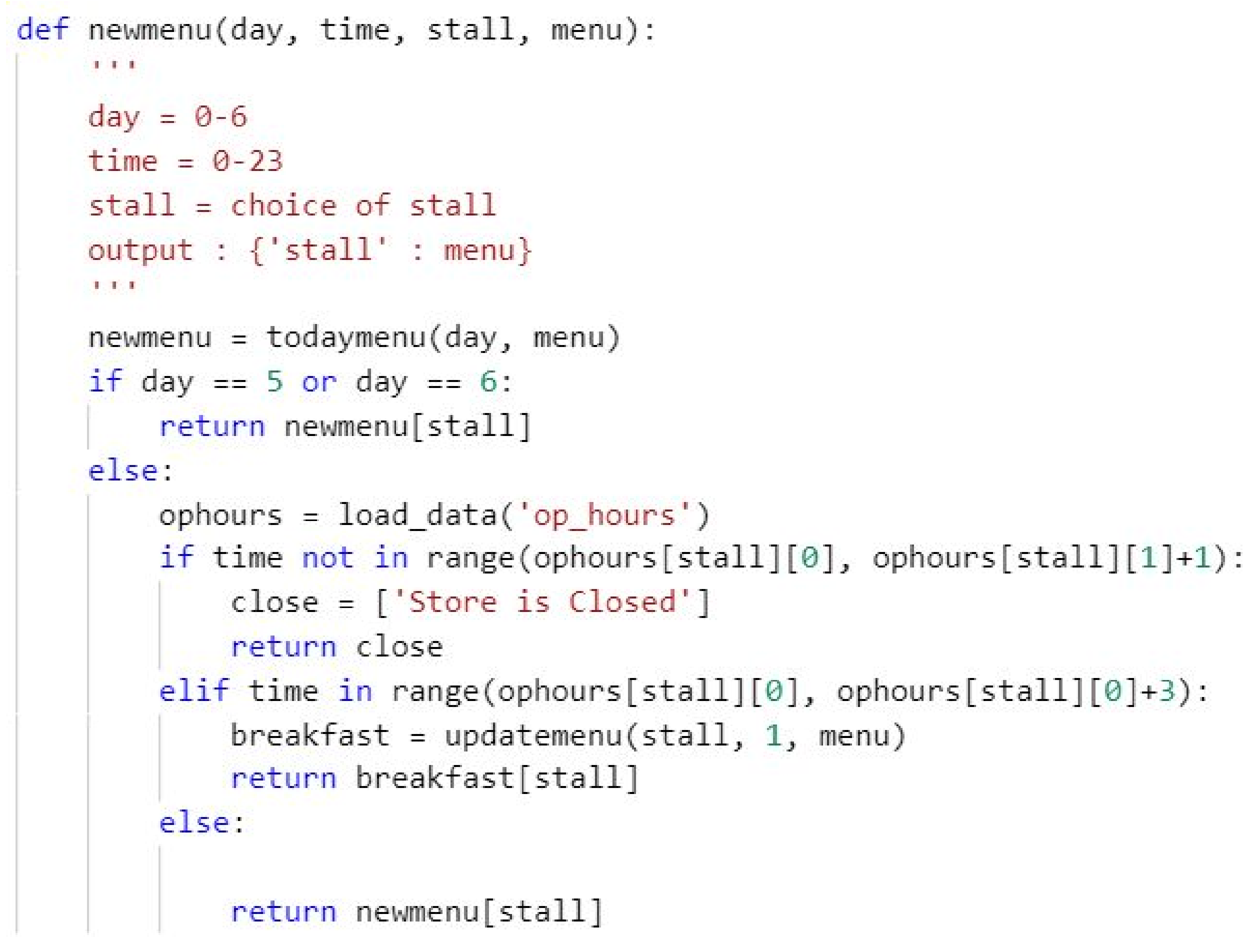
```python
#Generate menu based on day
def todaymenu(weekday, menu):
        #Special Menu Days
        menu = load_data('default_menu')
        keydays = [2,4,5,6]
        if weekday in keydays:
            if weekday == 5 or weekday == 6:
                menu = {stall:['Store is Closed'] for (stall,menu) in menu.items()}
                return menu
            elif weekday == 2:
                menu = updatemenu('noodles', 2, menu)
                return menu
            elif weekday == 4:
                menu = updatemenu('mala', 2, menu)
                return menu
        else:
            return menu
```

Figure 7: todaymenu() function

The ‘case’ is stored in the variable keydays. Depending on the day passed into the function, a menu based on that day is generated.

#HaoCheng & Jasper#

6. newmenu(day, time, stall):



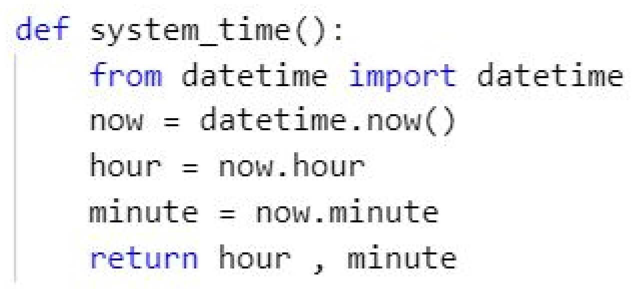
```
def newmenu(day, time, stall, menu):
    '''
    day = 0-6
    time = 0-23
    stall = choice of stall
    output : {'stall' : menu}
    '''
    newmenu = todaymenu(day, menu)
    if day == 5 or day == 6:
        return newmenu[stall]
    else:
        ophours = load_data('op_hours')
        if time not in range(ophours[stall][0], ophours[stall][1]+1):
            close = ['Store is Closed']
            return close
        elif time in range(ophours[stall][0], ophours[stall][0]+3):
            breakfast = updatemenu(stall, 1, menu)
            return breakfast[stall]
        else:

            return newmenu[stall]
```

Figure 8: newmenu() function

This function takes in day, time and stall name as argument, compare time parameter with the stall's operating time to determine whether it is open, followed by determining whether it is breakfast. Else, it will return the specific menu on that day.

#Haocheng#

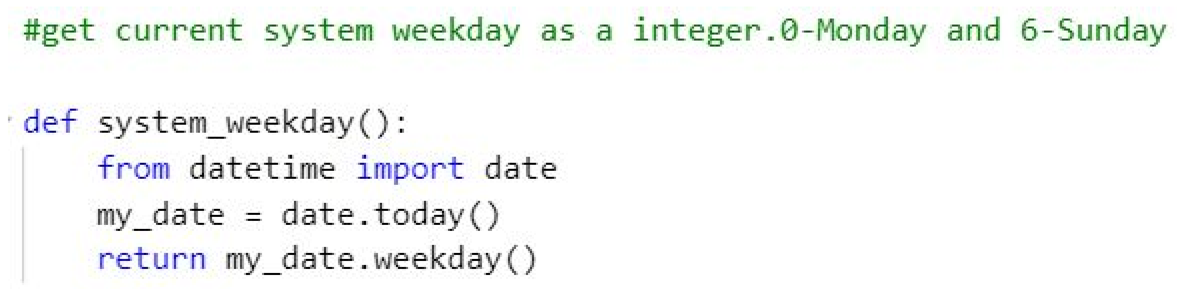
7. system_time()



```python
def system_time():
    from datetime import datetime
    now = datetime.now()
    hour = now.hour
    minute = now.minute
    return hour , minute
```

Figure 9: system_time() function

8. system_weekday()

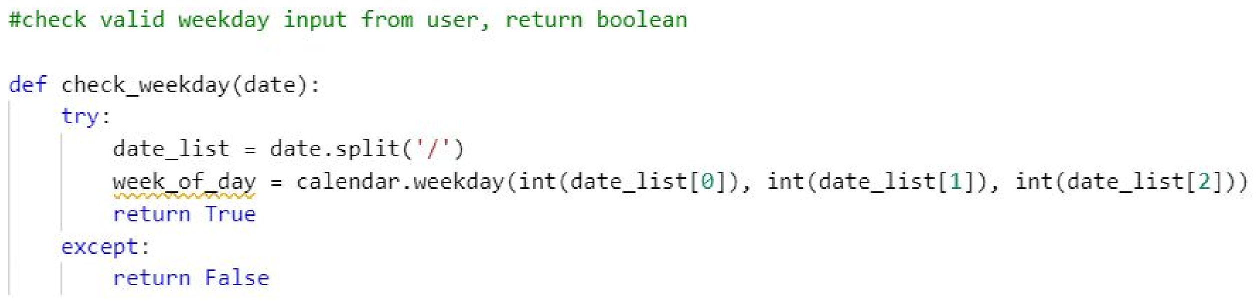


```python
#get current system weekday as a integer.0-Monday and 6-Sunday

def system_weekday():
    from datetime import date
    my_date = date.today()
    return my_date.weekday()
```

Figure 10: system_weekday() function

These two functions import python’s datetime module in order to access the date and time from the system, and return day and time respectively.

9. check_weekday()

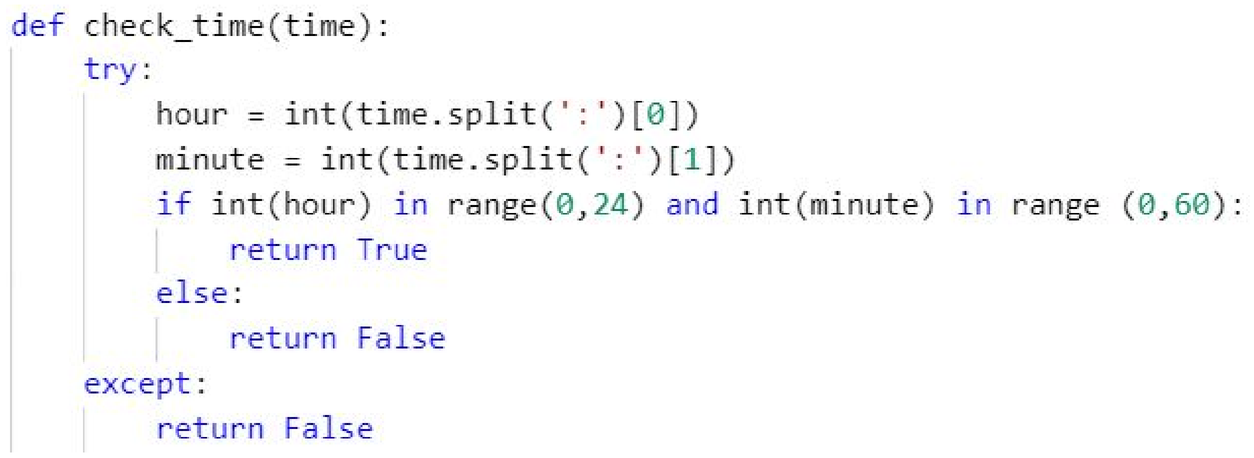


```
#check valid weekday input from user, return boolean

def check_weekday(date):
    try:
        date_list = date.split('/')
        week_of_day = calendar.weekday(int(date_list[0]), int(date_list[1]), int(date_list[2]))
        return True
    except:
        return False
```

Figure 11: check_weekday() function

This function takes in user's date input and use try/except to check for the validity of the input by passing them into the calendar module. Only correct date input will return True, so that the user_weekday() function will run.

10. check_time()



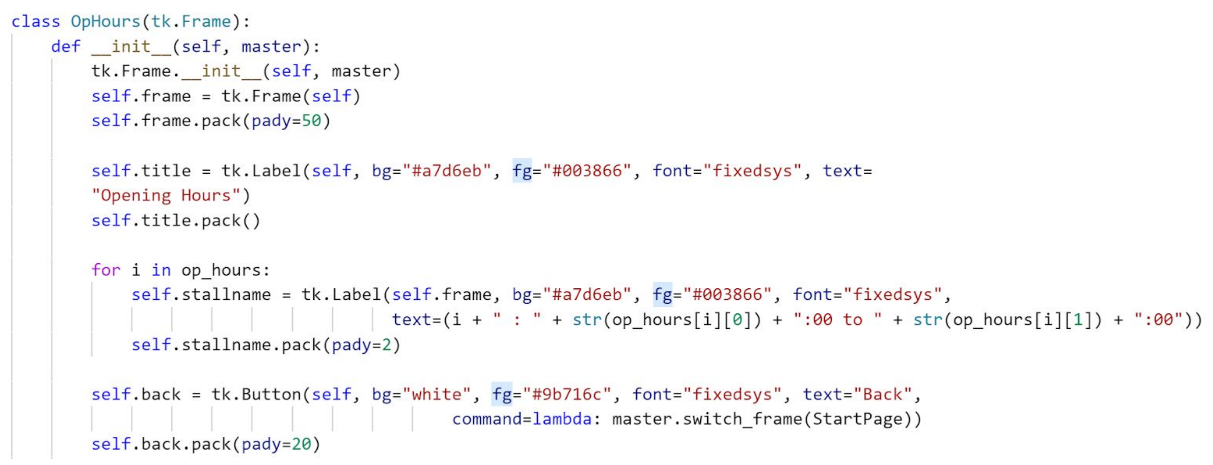
```
def check_time(time):
    try:
        hour = int(time.split(':')[0])
        minute = int(time.split(':')[1])
        if int(hour) in range(0,24) and int(minute) in range (0,60):
            return True
        else:
            return False
    except:
        return False
```

Figure 12: check_time() function

This function takes in user's time input. A try block is used to detect any syntax errors such as entering alphabet instead of numbers, and an if/else block to check any logic error such as inputting hour>23 or minute>59

#Khin#

View operating hours

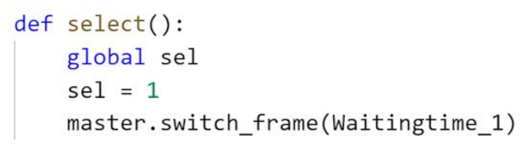


```
class OpHours(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        self.frame = tk.Frame(self)
        self.frame.pack(pady=50)

        self.title = tk.Label(self, bg="#a7d6eb", fg="#003866", font="fixedsys", text=
        "Opening Hours")
        self.title.pack()

        for i in op_hours:
            self.stallname = tk.Label(self.frame, bg="#a7d6eb", fg="#003866", font="fixedsys",
                                      text=(i + " : " + str(op_hours[i][0]) + ":00 to " + str(op_hours[i][1]) + ":00"))
            self.stallname.pack(pady=2)

        self.back = tk.Button(self, bg="white", fg="#9b716c", font="fixedsys", text="Back",
                              command=lambda: master.switch_frame(StartPage))
        self.back.pack(pady=20)
```

Figure 13: OpHours Class

This code shows how opening hours for stalls from a dictionary in database.py are loaded onto a frame. The opening hours are then formatted to follow the HH:MM style.

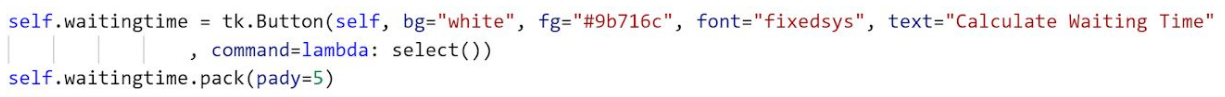
Estimated waiting time



```
def select():
    global sel
    sel = 1
    master.switch_frame(Waitingtime_1)
```

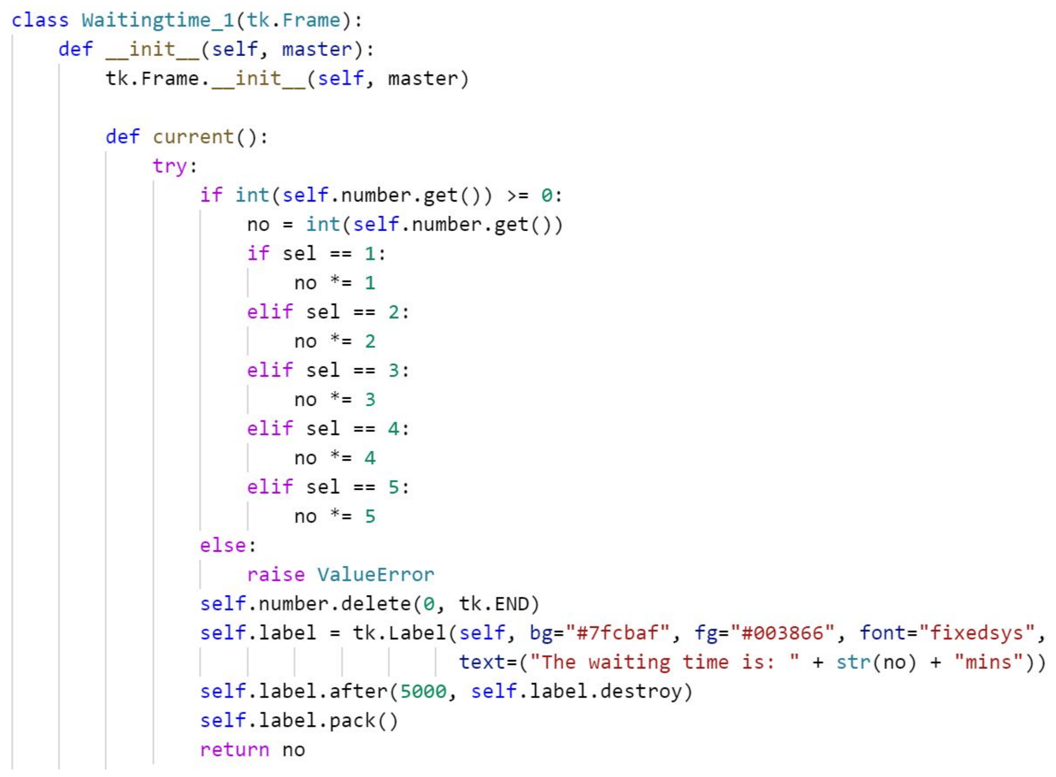
Figure 14: select function

“select” function placed in menu display of every stall allows program to track the origin route and calculate different waiting time for each stall.



```
self.waitingtime = tk.Button(self, bg="white", fg="#9b716c", font="fixedsys", text="Calculate Waiting Time"
                , command=lambda: select())
self.waitingtime.pack(pady=5)
```

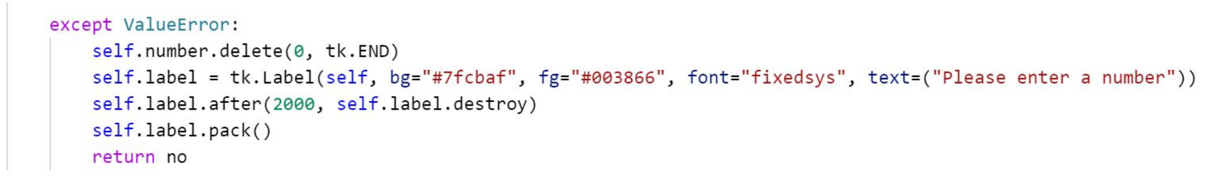
Figure 15: Button that calls Waitingtime_1 class



```
class Waitingtime_1(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)

        def current():
            try:
                if int(self.number.get()) >= 0:
                    no = int(self.number.get())
                    if sel == 1:
                        no *= 1
                    elif sel == 2:
                        no *= 2
                    elif sel == 3:
                        no *= 3
                    elif sel == 4:
                        no *= 4
                    elif sel == 5:
                        no *= 5
                else:
                    raise ValueError
                self.number.delete(0, tk.END)
                self.label = tk.Label(self, bg="#7fcbaf", fg="#003866", font="fixedsys",
                                      text=("The waiting time is: " + str(no) + "mins"))
                self.label.after(5000, self.label.destroy)
                self.label.pack()
                return no
```

Figure 16: Waitingtime_1 class part 1

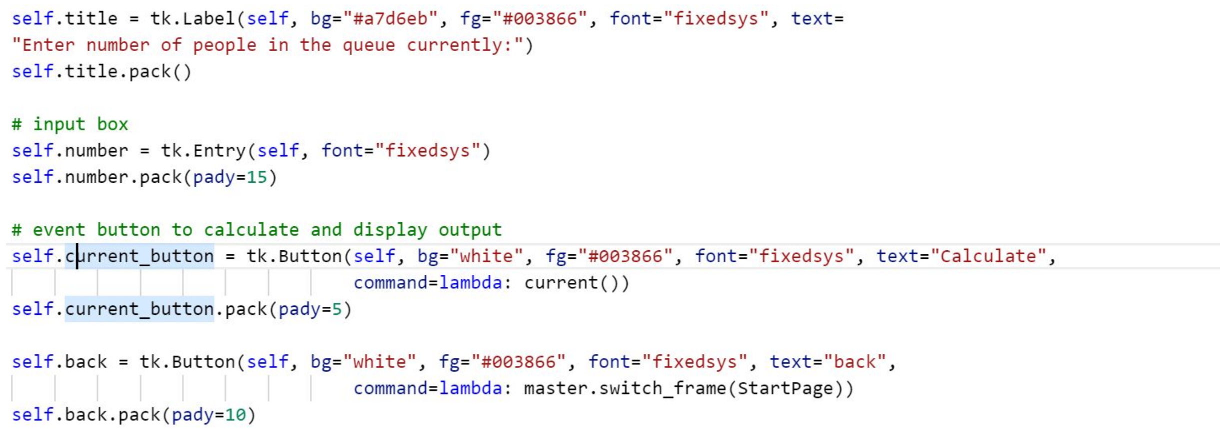
"Current" function in the Waitingtime_1 Class. The function checks that the input typed for is a positive integer and origin stall route in order to provide different waiting times per person for different stalls.



```
            except ValueError:
                self.number.delete(0, tk.END)
                self.label = tk.Label(self, bg="#7fcbaf", fg="#003866", font="fixedsys", text=("Please enter a number"))
                self.label.after(2000, self.label.destroy)
                self.label.pack()
                return no
```

Figure 17 : Waitingtime_1 class part 2

"except" catches any input that are not an integer and prompts the user to re-enter input.

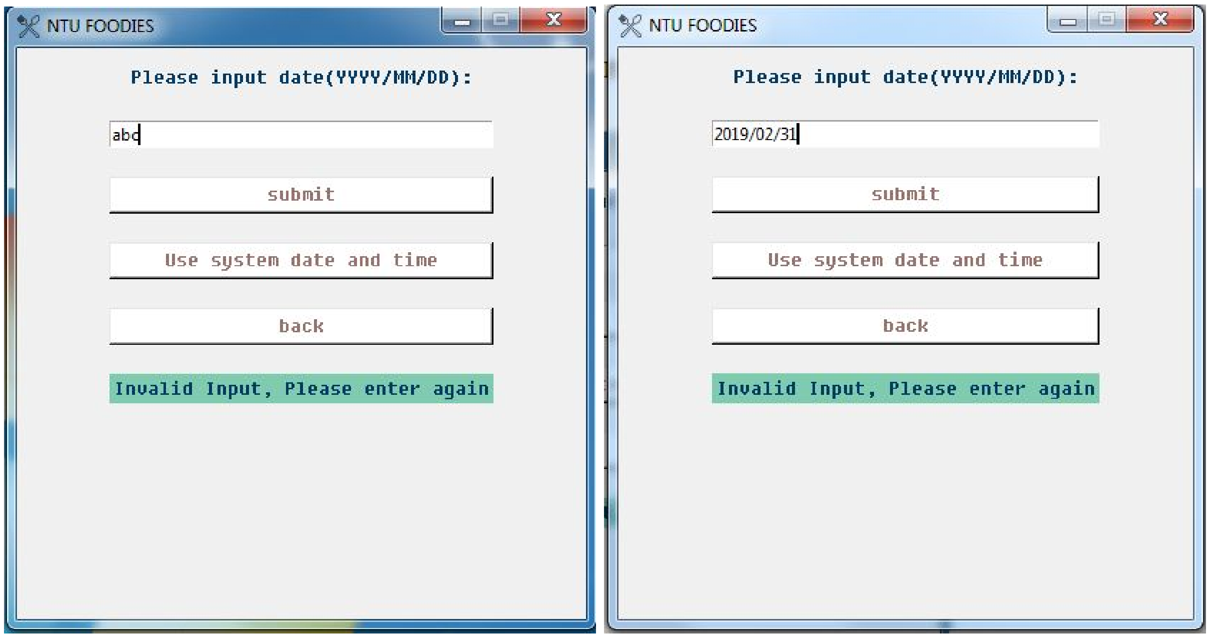


```
self.title = tk.Label(self, bg="#a7d6eb", fg="#003866", font="fixedsys", text=
"Enter number of people in the queue currently:")
self.title.pack()

# input box
self.number = tk.Entry(self, font="fixedsys")
self.number.pack(pady=15)

# event button to calculate and display output
self.current_button = tk.Button(self, bg="white", fg="#003866", font="fixedsys", text="Calculate",
                                command=lambda: current())
self.current_button.pack(pady=5)

self.back = tk.Button(self, bg="white", fg="#003866", font="fixedsys", text="back",
                                command=lambda: master.switch_frame(StartPage))
self.back.pack(pady=10)
```

Figure 18 : Waitingtime_1 class part 3

The main part of WaitingTime_1 Class which includes the widgets (textbox and buttons)

Program testing



Please input date(YYYY/MM/DD):

abc

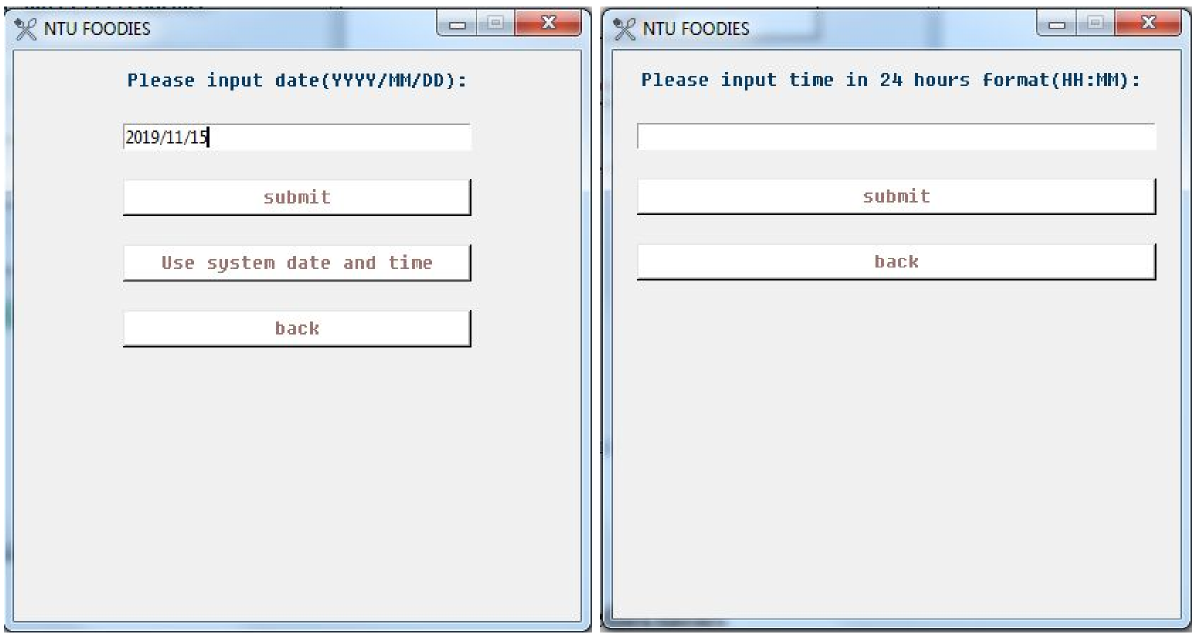
Invalid Input, Please enter again

Please input date(YYYY/MM/DD):

2019/02/31

Invalid Input, Please enter again

Figure 19: Invalid date input from user

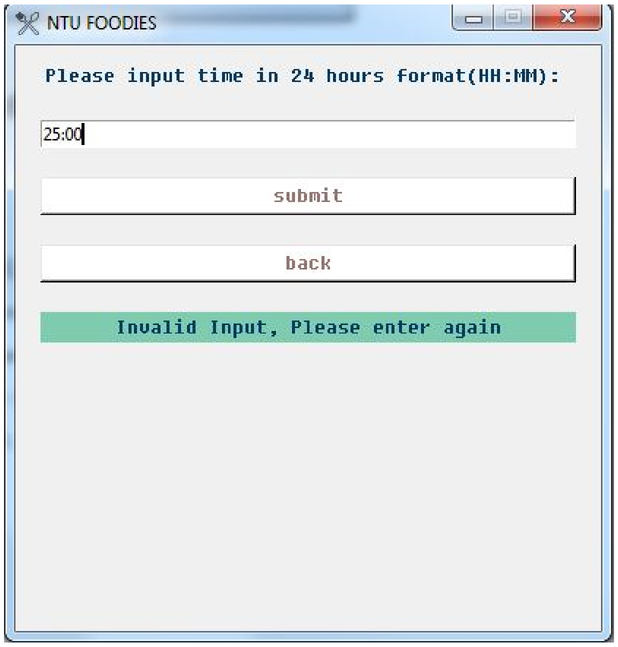


Please input date(YYYY/MM/DD):

2019/11/15

Please input time in 24 hours format(HH:MM):

Figure 20: Valid date input lead to time input page

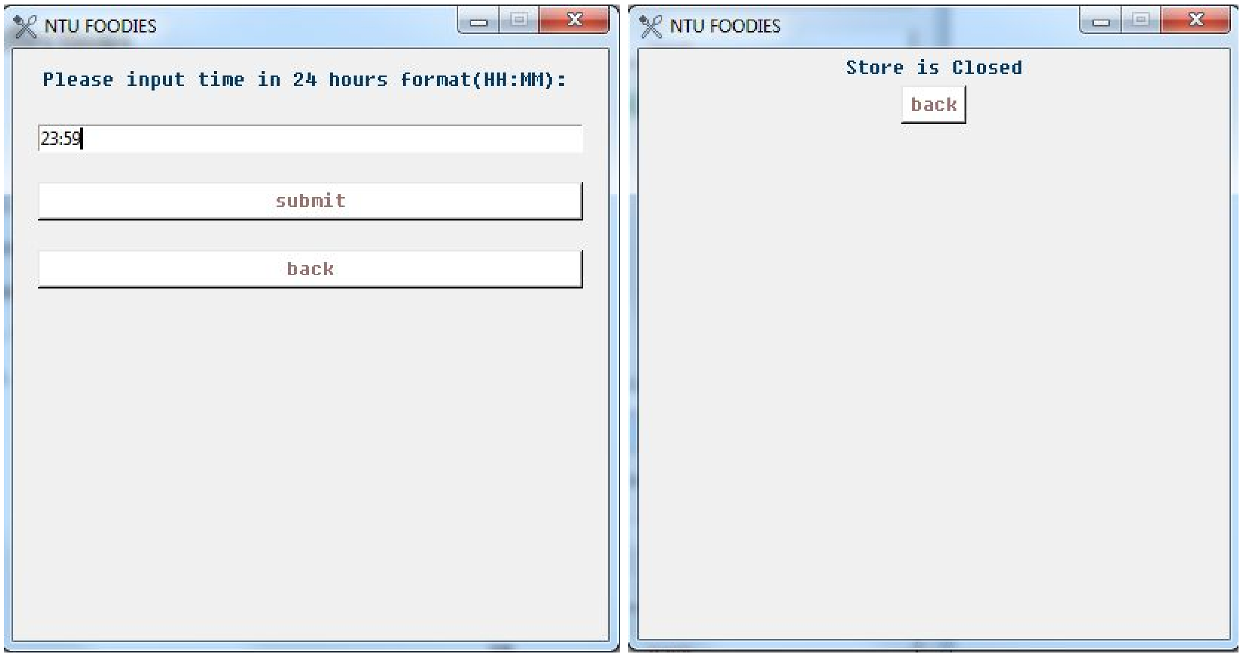

NTU FOODIES

Please input time in 24 hours format(HH:MM):

25:00

Invalid Input, Please enter again

Figure 21: Invalid time input and error message


NTU FOODIES

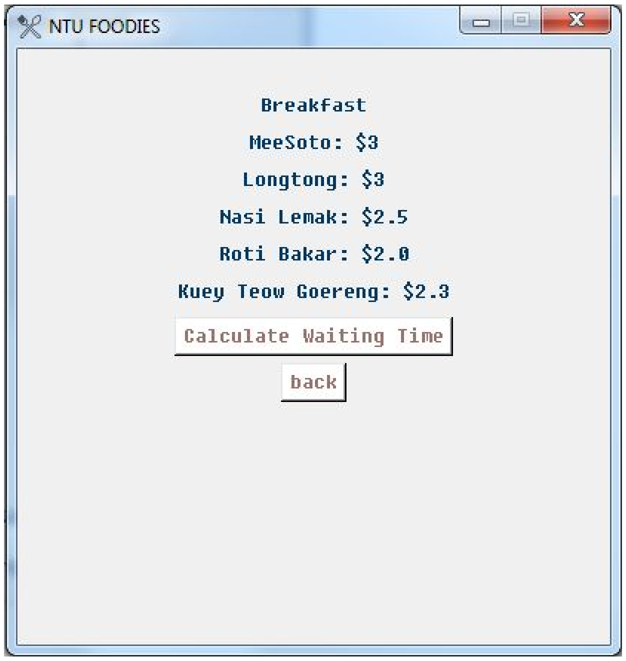
Please input time in 24 hours format(HH:MM):

23:59

NTU FOODIES

Store is Closed

Figure 22: input time outside operating time



Breakfast

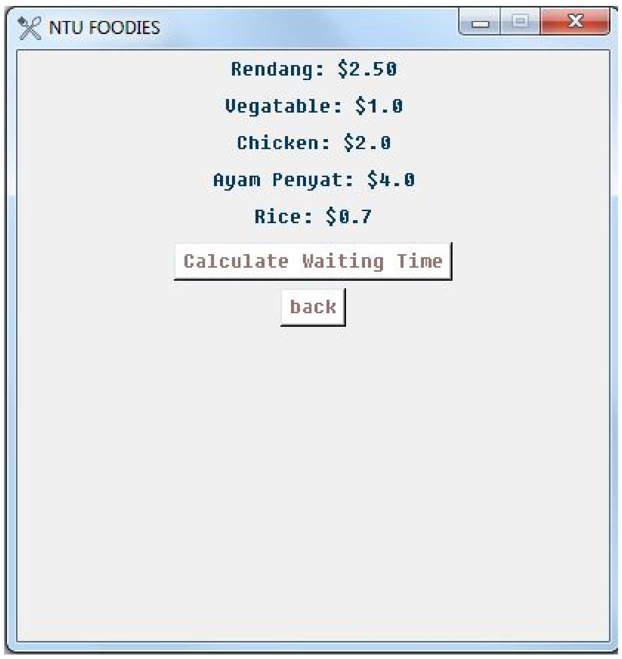
MeeSoto: $3

Longtong: $3

Nasi Lemak: $2.5

Roti Bakar: $2.0

Kuey Teow Goereng: $2.3



Rendang: $2.50

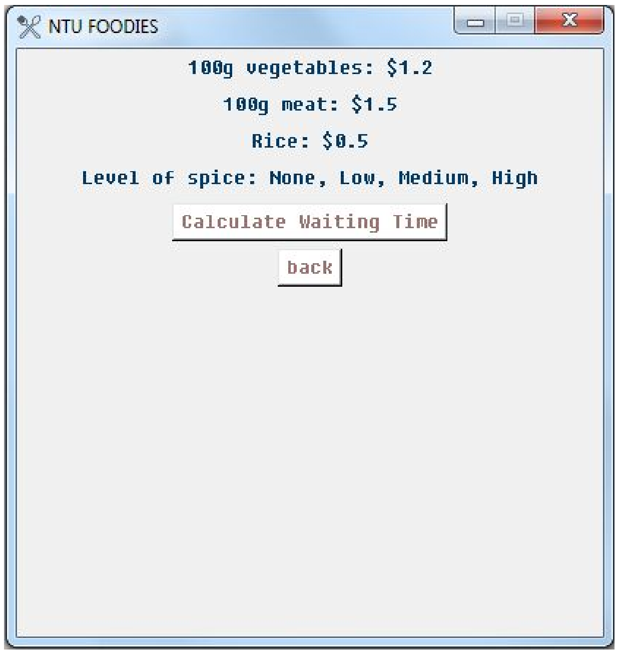
Vegatable: $1.0

Chicken: $2.0

Ayam Penyat: $4.0

Rice: $0.7

Figure 23: Special menu: breakfast

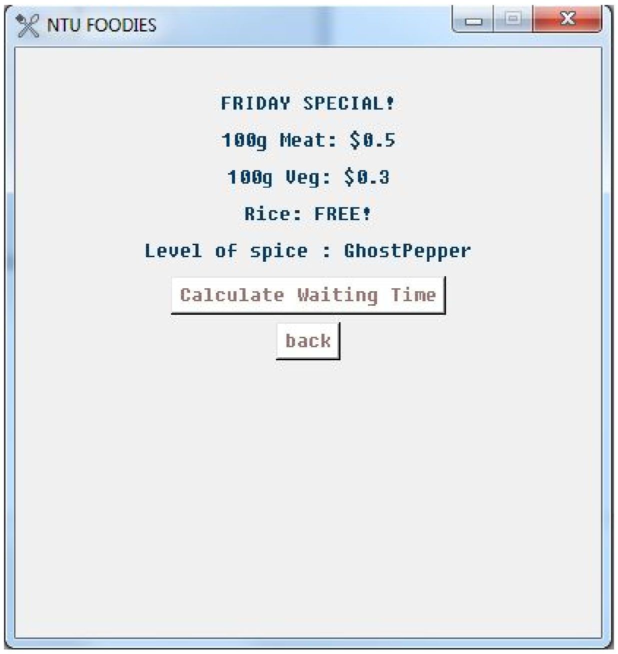


100g vegetables: $1.2

100g meat: $1.5

Rice: $0.5

Level of spice: None, Low, Medium, High



FRIDAY SPECIAL!

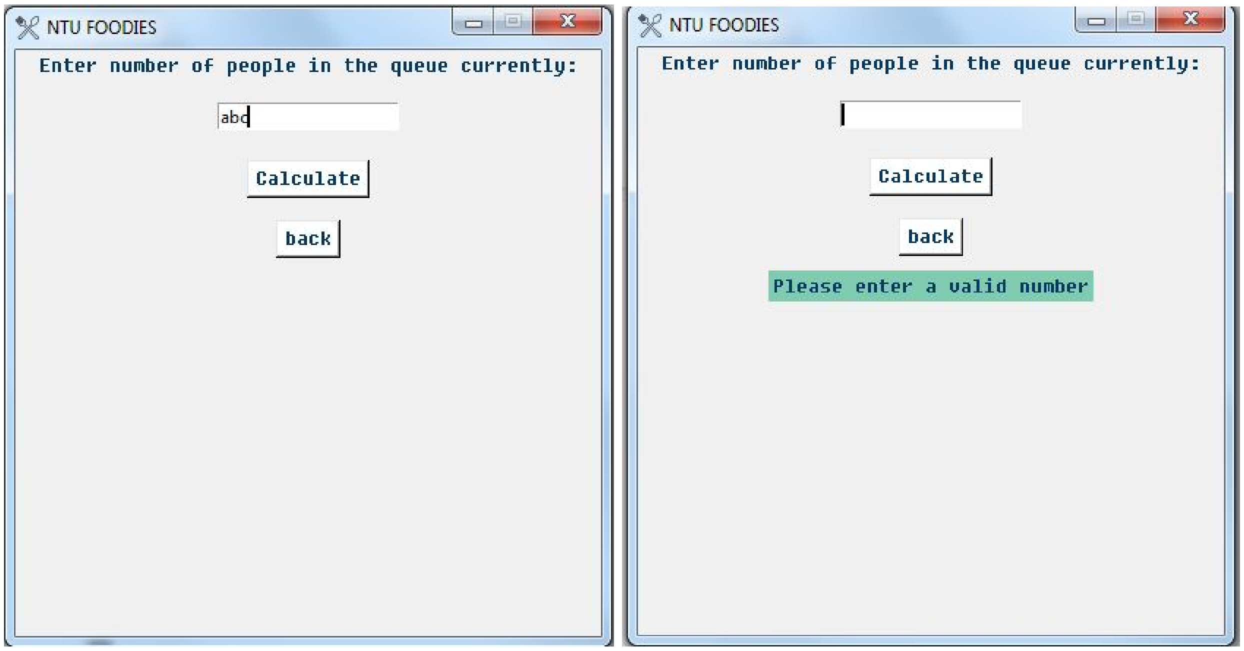
100g Meat: $0.5

100g Veg: $0.3

Rice: FREE!

Level of spice : GhostPepper

Figure 24: Special menu: Friday


NTU FOODIES

Enter number of people in the queue currently:

abc

NTU FOODIES

Enter number of people in the queue currently:

Please enter a valid number

Figure 25 : Error show when non-integer, negative integer or fractions are input

Reflections

Khin

The team decided to go with Tkinter for GUI as it seemed to be the easiest to learn and have the most resources out of all the choices. However, the journey to understand and use it was not a very smooth one. Also, the concept of classes, objects and functions working together was new and it was a little confusing to learn on the job. Thankfully, I had teammates, Jasper and Haocheng, to go through this together with. Although our final project may not look the best, I am proud of what we came up with.

Through this project, we turned to many online sources to help us out. This process made us more independent. However, I feel that there should be a LAMS/lecture session focused on Graphical User Interface creation.

Haocheng

For my part, the main difficulty encountered was to pass variables between different classes, since date and time variables are acquired in one class while displaying menu page is in another class. Eventually, I solved this problem by declaring global variables within one class and pass these values into another class. I learnt this idea during face-to-face lectures, which proves to be extremely helpful. Other than that, I was at first confused about assigning multiple functions to a single button and solved it by defining a function that initiates other functions, after doing some online research.

Despite the difficulties, I have used various approaches to solve the problems encountered. This experience of independent thinking and problem-solving cannot be achieved outside a hands-on project. I have learnt various approaches to solve a difficult problem, such as attempting with different inputs during debugging and doing online research. Most importantly, this first CS project significantly improves my enthusiasm toward computing. In general, I am satisfied with what we have come up with and I am looking forward to the next computing project.

Jasper

Planning the program(either by flowchart or otherwise) before coding was an important learning point for myself. With the whole program having so many features, it is easy to be overwhelmed. A well-planned flowchart allows the breakdown of the entire program into manageable sections, makes coding smoother as well as allow proper distribution of workload between team members.

An issue that I struggled with was namespace, because I mainly created functions to manipulate the global menu, I often had unwanted results with the functions. However, after the "Advanced Functions" lecture and more practice with the code, I feel that I got a better understanding of the whole topic in general.

Tkinter was also a huge challenge, tutorials online mainly focused on object oriented programming for the GUI generation. Hence that was what we tried to do, however without firm understanding in concepts of OOP and Classes, working with tkinter proved highly difficult. This is an area I will strive to understand moving forward in my programming journey.