

NANYANG
TECHNOLOGICAL
UNIVERSITY

MINI PROJECT

CE1003/CZ1003

Real-time Canteen Information System Report

LIU QINGYI (U1921143C)

NICHOLAS EE (U1620678B)

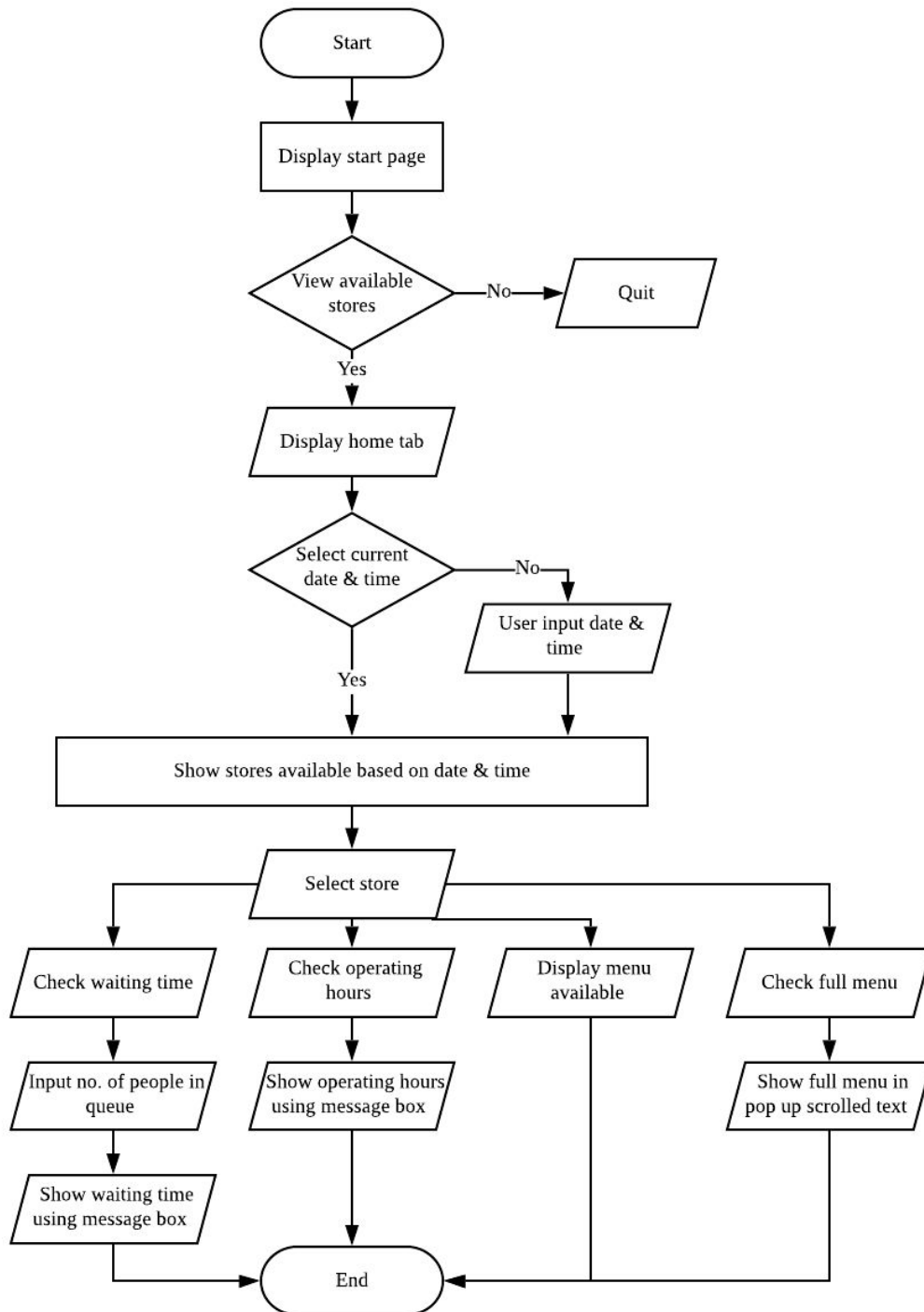
SHAUN ONG (U1921868K)

SCHOOL OF COMPUTER SCIENCE and ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Content:

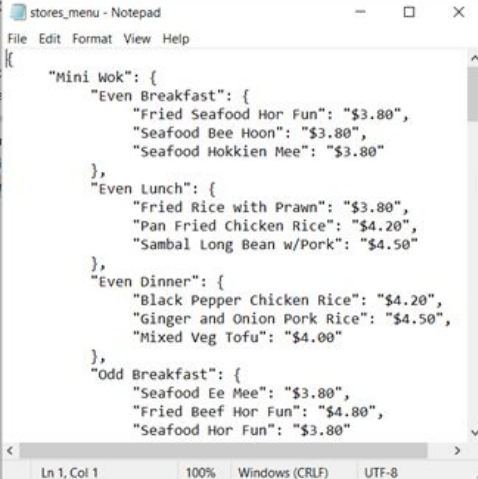
- 1. Algorithm Design (Shaun)**
 - 1.1 Flow chart**
 - 1.2 Json file handling and functions**
 - 1.3 GUI functions**
- 2. Program testing (Qing Yi)**
 - 2.1 External modules for error handling**
 - 2.2 Try/except statements for error handling**
- 3. Reflection (Nicholas)**
 - 3.1 Difficulties encountered**
 - 3.2 Ways to conquer**
 - 3.3 Knowledge learnt from this course**
 - 3.4 Further improvement and suggestions**
- 4. References**

1.1 Flow chart

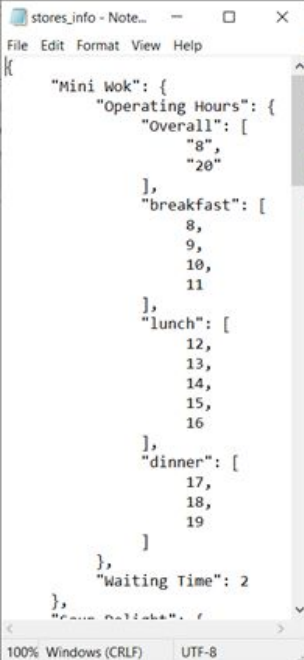
1.2 Json file handling and functions

1.2.1 Data Storage

We used json files to store the data of store menus and store information.



```
{
  "Mini Wok": {
    "Even Breakfast": {
      "Fried Seafood Hor Fun": "$3.80",
      "Seafood Bee Hoon": "$3.80",
      "Seafood Hokkien Mee": "$3.80"
    },
    "Even Lunch": {
      "Fried Rice with Prawn": "$3.80",
      "Pan Fried Chicken Rice": "$4.20",
      "Sambal Long Bean w/Pork": "$4.50"
    },
    "Even Dinner": {
      "Black Pepper Chicken Rice": "$4.20",
      "Ginger and Onion Pork Rice": "$4.50",
      "Mixed Veg Tofu": "$4.00"
    },
    "Odd Breakfast": {
      "Seafood Ee Mee": "$3.80",
      "Fried Beef Hor Fun": "$4.80",
      "Seafood Hor Fun": "$3.80"
    }
  }
}
```



```
{
  "Mini Wok": {
    "Operating Hours": {
      "Overall": [
        "8",
        "20"
      ],
      "breakfast": [
        8,
        9,
        10,
        11
      ],
      "lunch": [
        12,
        13,
        14,
        15,
        16
      ],
      "dinner": [
        17,
        18,
        19
      ]
    },
    "Waiting Time": 2
  }
}
```

1.2.2 `def menu_by_time(store,day,hour):`

```
# Returns menu based on specific store, preferred date and time

def menu_by_time(store,day,hour):
    breakfast=stores_info[store]["Operating Hours"]["breakfast"]
    lunch=stores_info[store]["Operating Hours"]["lunch"]
    dinner=stores_info[store]["Operating Hours"]["dinner"]

    if day in odd_day and hour in breakfast:
        food=stores_menu[store]["Odd Breakfast"]
    elif day in odd_day and hour in lunch:
        food=stores_menu[store]["Odd Lunch"]
    elif day in odd_day and hour in dinner:
        food=stores_menu[store]["Odd Dinner"]
    elif day in even_day and hour in breakfast:
        food=stores_menu[store]["Even Breakfast"]
    elif day in even_day and hour in lunch:
        food=stores_menu[store]["Even Lunch"]
    elif day in even_day and hour in dinner:
        food=stores_menu[store]["Even Dinner"]
    else:
        return None,None

    if len(food) !=0:
        count=0
        list_of_food=""
        list_of_price=""
        for key,value in food.items():
            count+=1
            key="{0}. {1}\n".format(count,key)
            value += "\n"
            list_of_food+=key
            list_of_price+=value
        return list_of_food,list_of_price
```

This function takes in day and hour values and return strings containing both the food and price which was extracted from the nested json data. The function returns the corresponding menu based on the store parameter that was passed in.

1.2.3 `def operating_hours(store):`

```
# Returns operating hours for a specific store

def operating_hours(store):
    time=stores_info[store]["Operating Hours"]["Overall"]
    start=datetime.strptime(time[0],"%H").strftime("%H%M")
    end=datetime.strptime(time[1],"%H").strftime("%H%M")
    return start,end
```

This function will extract and return start and end hours of operating hours based on the store parameter.

1.2.4 `def waiting_time(store,people):`

```
# Returns waiting time for a specific store based on the number of people in the queue

def waiting_time(store,people):
    time=stores_info[store]["Waiting Time"]*people
    return "Waiting time is approximately: "+str(time)+" minutes!"
```

This function extracts the average waiting time for each person based on the store parameter and multiply this value with the number of people which was passed in as a parameter. The end result is concatenated with a string format and returned.

1.2.5 `def store_menu(store):`

```
# Displays all store menus or returns all menu for a particular store

def store_menu(store):
    if store=="All":
        for store in stores_menu:
            display_store_name(store)
            count=0
            list_of_food=""
            list_of_price=""
            menu=stores_menu[store]
            for day_hour in menu.values():
                for key,value in day_hour.items():
                    count+=1
                    item="{:2}. {:40}{ }\n".format(count,key,value)
                    list_of_food+=item
            print(list_of_food,'\n')
    else:
        menu=stores_menu[store]
        count=0
        list_of_food=""
        list_of_price=""
        for day_hour in menu.values():
            for key,value in day_hour.items():
                count+=1
                item="{:2}. {:40}{ }\n".format(count,key,value)
                list_of_food+=item
        return list_of_food
```

This function extracts the full menu of all stores and displays them on shell. Alternatively, it returns the string for the full menu of each store.

1.3 GUI functions

1.3.1 def now_time():

```
def now_time():
    self.time_input=False
    datetime_entry_frame.place_forget()
    home_datetime_input.place_forget()
    home_clock.place(relx = 0.5, rely = 0.35, anchor="n")
    messagebox.showinfo(title="Success!", message = "List of available stores has been updated based on current date and time.")
```

The function sets clock to current date and time and sets time_input as False.

1.3.2 def enter_time():

```
def enter_time():
    date=str(date_cal_entry.get_date())
    year, month, day = (x for x in date.split("-"))

    try:
        time = time_entry.get()
        hour, mins = (int(x) for x in time.split(":"))
        if 0 <= hour <= 23 and 0 <= mins <= 59:
            self.time_input = True
            hour,mins = str(hour), str(mins)
            date_input = datetime.strptime(year+month+day+hour+mins,"%Y%m%d%H%M")
            self.day_input = date_input.strftime("%A")
            self.hour_input = date_input.strftime("%H")
            label = date_input.strftime("%A, %d %B, %Y, %H:%M:%S")
            home_datetime_input.config(text = label)
            home_clock.place_forget()
            home_datetime_input.place(relx = 0.5, rely = 0.35, anchor="n")
            messagebox.showinfo(title="Success!", message = "List of available stores has been updated based on date and time input.")
        else:
            messagebox.showinfo(title="Invalid Input!", message = "Hour must be within 0 - 23\nMinutes must be within 0 - 59")
    except ValueError:
        messagebox.showinfo(title="Invalid Input!", message = "Please enter time in HH:MM format.\nHour and mins must be an integer. Eg. 12:00")
```

The function obtains user input date and time from the entry box and sets time_input as True.

1.3.3 def store_menus():

```
#update window based on current or user input date & time
def store_menus():
    index=tab_control.index(tab_control.select())
    store={1:"Mini Wok",2:"Soup Delight",3:"McDonald's",4:"UmiSushi",5:"Fun World Cafe"}

    #set day and hour based on current or user input
    if self.time_input == False:
        day = datetime.now().strftime("%A")
        hour = int(datetime.now().strftime("%H"))
    else:
        day = self.day_input
        hour = int(self.hour_input)

    #show and hide stores tabs based on operating hours of stores
    modes = [
        ("Mini Wok",mini_wok_tab),
        ("Soup Delight",soup_delight_tab),
        ("McDonald's",macdonalds_tab),
        ("UmiSushi",umi_sushi_tab),
        ("Fun World Cafe",fun_world_cafe_tab)
    ]
    for stall,tab in modes:
        start, end = operating_hours(stall)
        start = start[:2]
        end = end[:2]
        if hour not in range(int(start),int(end)) or day == "Sunday":
            tab_control.hide(tab)
        else:
            tab_control.add(tab)

    #change home tab background based on time
    if hour in range(0,12):
        photo = tk.PhotoImage(file = "Pictures/morning_background.gif")
        home_background.image=photo
        home_background.itemconfigure(home_background_image,image=photo)
    elif hour in range(12,18):
        photo = tk.PhotoImage(file = "Pictures/afternoon_background.gif")
        home_background.image=photo
        home_background.itemconfigure(home_background_image,image=photo)
    elif hour in range(18,24):
        photo = tk.PhotoImage(file = "Pictures/evening_background.gif")
        home_background.image=photo
        home_background.itemconfigure(home_background_image,image=photo)

    #change store sign based on time
    if hour in range(7,21):
        photo = tk.PhotoImage(file = "Pictures/Open.gif")
        home_sign.image = photo
        home_sign.itemconfigure(home_sign_image, image=photo)
    else:
        photo = tk.PhotoImage(file = "Pictures/Closed.gif")
        home_sign.image = photo
        home_sign.itemconfigure(home_sign_image, image=photo)

    #change menu items based on date & time
    if index in range(1,6):
        food,price = menu_by_time(store[index],day,hour)
        menu.itemconfigure(food_display, text=food)
        menu.itemconfigure(price_display, text=price)

    self.after(100,store_menus)
```

As different stores have different data for store menus, operating hours and waiting time, we will retrieve the index of the store tab that user is currently viewing. The index is then used to retrieve the data from a json file specific to the store.

The function updates the GUI based on current or user input date & time. When the program starts, time_input will be False. When time_input is False, variable day and hour will be based on current date and time. When time_input is True, variable day and hour will be based on user's input. The store tabs will be hidden or added and the menu will be updated based on day and hour.

1.3.4 def operating_hours_msgbox():

```
def operating_hours_msgbox():
    stores={1:"Mini Wok",2:"Soup Delight",3:"McDonald's",4:"UmiSushi",5:"Fun World Cafe"}
    index=tab_control.index(tab_control.select())
    start, end = operating_hours(stores[index])
    messagebox.showinfo(title=stores[index]+" Operating Hours", message = "Monday to Saturday: "\
        +start+" - "+end)
```

The function uses the store index when invoking the operating_hours function from function_list to retrieve the store's operating hours, which is then displayed in a message box.

1.3.5 def waiting_time_msgbox():

```
def waiting_time_msgbox():
    stores={1:"Mini Wok",2:"Soup Delight",3:"McDonald's",4:"UmiSushi",5:"Fun World Cafe"}
    index=tab_control.index(tab_control.select())
    try:
        ppl = int(no_of_ppl.get())
        if 0<=ppl<=100:
            msg = waiting_time(stores[index],ppl)
            messagebox.showinfo(title= stores[index]+" Waiting Time", message = msg)
        else:
            messagebox.showinfo(title="Invalid Input!", message = "Please enter a positive integer (0-100).")
    except ValueError:
        messagebox.showinfo(title="Invalid Input!", message = "Please enter a positive integer (0-100).")
```

The function uses the store index when invoking the waiting_time function from function_list to retrieve the store's total waiting time based on the number of people which was retrieved from the entry box. The result is displayed in a message box.

1.3.6 def full_menu():

```
def full_menu():
    stores={1:"Mini Wok",2:"Soup Delight",3:"McDonald's",4:"UmiSushi",5:"Fun World Cafe"}
    index=tab_control.index(tab_control.select())
    top = tk.Toplevel()
    top.title(stores[index]+" Full Menu")
    top.resizable(width=False, height=False)

    full_menu_text = scrolledtext.ScrolledText(top, width=50, height=10)
    full_menu_text.pack(fill='both', expand="true")
    full_menu_text.insert(tk.END, store_menu(stores[index]))
    full_menu_text.config(state='disabled')
```

The function uses the store index when invoking the store_menu function from function_list to retrieve the store's full menu, which is then displayed in a scrolledtext widget in a popup window.

Chapter 2 Program testing (Qing Yi)

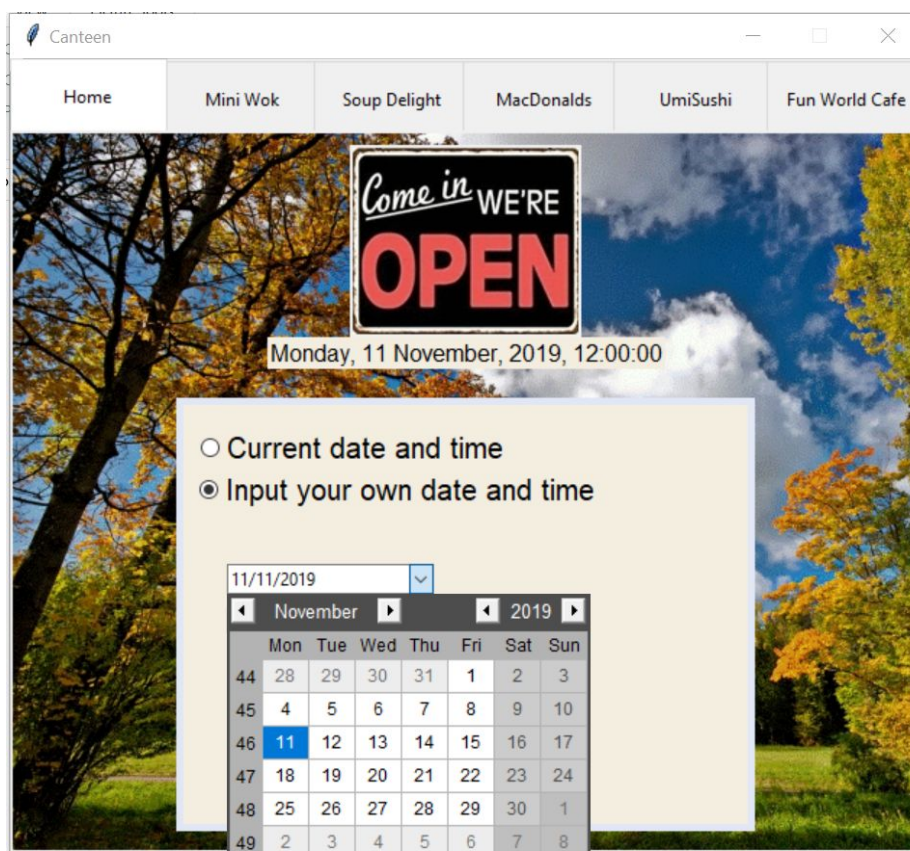
For the program testing, we used external modules and try/except statements to do error handling.

2.1 External modules for error handling:

We used tkcalendar for the error handling of date input.

```
#date & time entry frame
datetime_entry_frame = tk.Frame(home_frame,bg="#F4EEE1")
date_cal_entry = DateEntry(datetime_entry_frame,width=20,date_pattern='dd/mm/yyyy')
date_cal_entry.place(rely=0.05,relx=0.1)
date_cal_entry.config(state="readonly")
```

In the `enter_time()` function, we used an external module, tkcalendar. By using DateEntry in tkcalendar, we set the state of the date_cal_entry frame to "readonly". Then, the user input will be received by calling `date_cal_entry.get_date()` in `enter_time()`. Hence, only valid date inputs will be accepted as user can only select date input by clicking on the date from the drop down calendar.



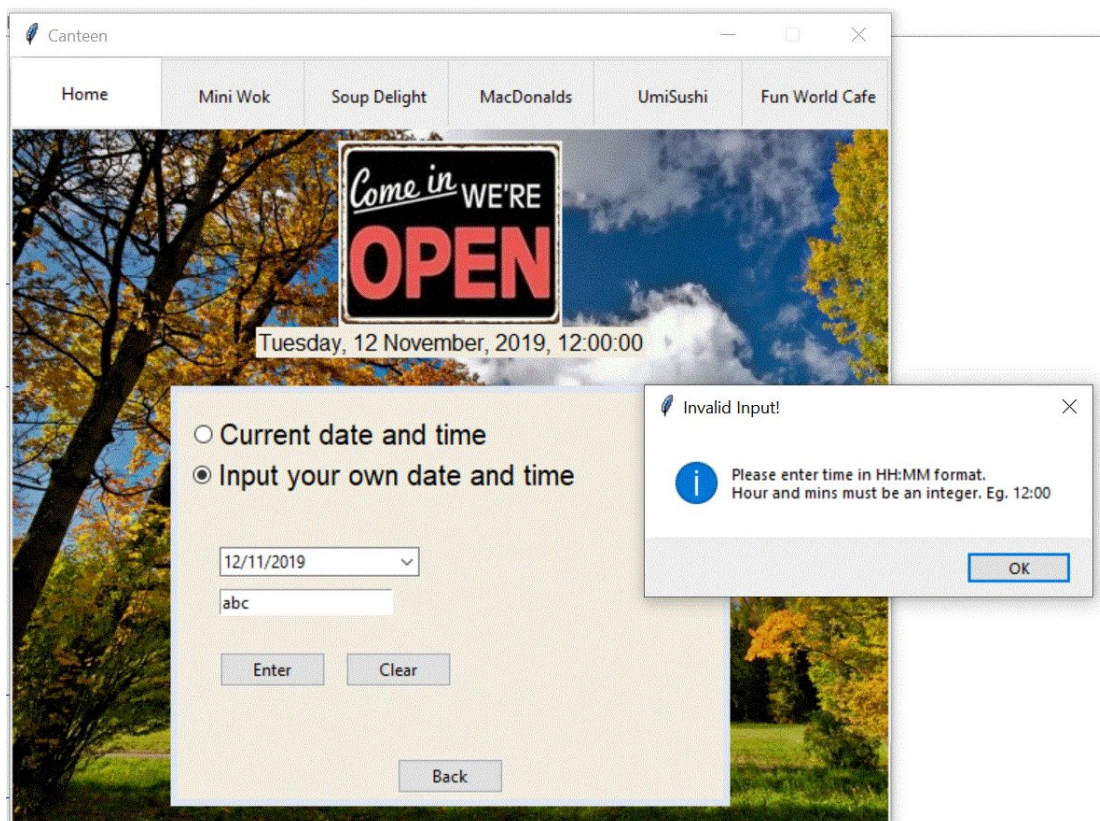
2.2 Try/except statement for error handling:

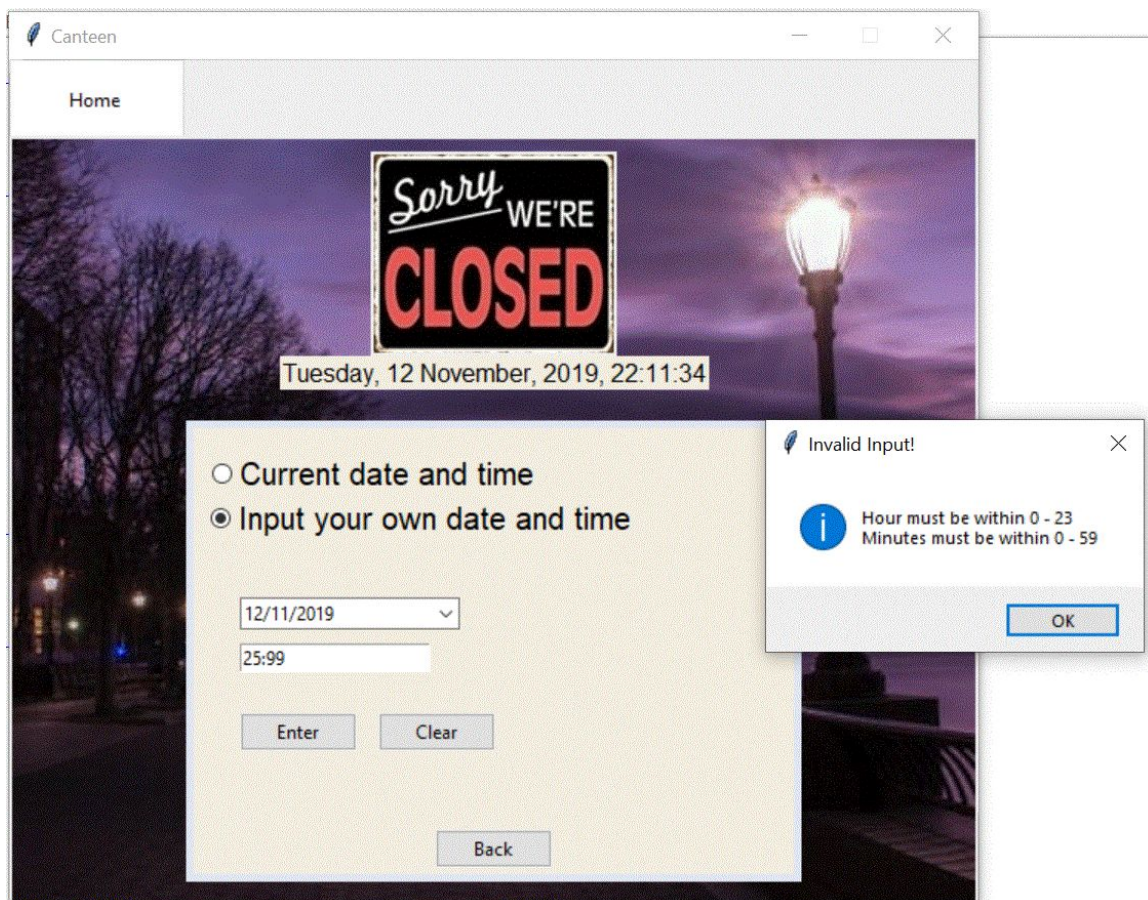
2.2.1 Time input error handling

```
def enter_time():
    date=str(date_cal_entry.get_date())
    year, month, day = (x for x in date.split("-"))

    try:
        time = time_entry.get()
        hour, mins = (int(x) for x in time.split(":"))
        if 0 <= hour <= 23 and 0 <= mins <= 59:
            self.time_input = True
            hour,mins = str(hour), str(mins)
            date_input = datetime.strptime(year+month+day+hour+mins,"%Y%m%d%H%M")
            self.day_input = date_input.strftime("%A")
            self.hour_input = date_input.strftime("%H")
            label = date_input.strftime("%A, %d %B, %Y, %H:%M:%S")
            home_datetime_input.config(text = label)
            home_clock.place_forget()
            home_datetime_input.place(relx = 0.5, rely = 0.35, anchor="n")
            messagebox.showinfo(title="Success!", message = "List of available stores has been updated based on date and time input.")
        else:
            messagebox.showinfo(title="Invalid Input!", message = "Hour must be within 0 - 23\nMinutes must be within 0 - 59")
    except ValueError:
        messagebox.showinfo(title="Invalid Input!", message = "Please enter time in HH:MM format.\nHour and mins must be an integer. Eg. 12:00")
```

The `enter_time()` function checks for valid time input. We used the `split(":")` method to ensure input is in the format "HH:MM". Improper or no assignment after split method will display an error message. Similarly, we used if/else statements to check for valid integer input. Invalid integer inputs prompts a different error message.

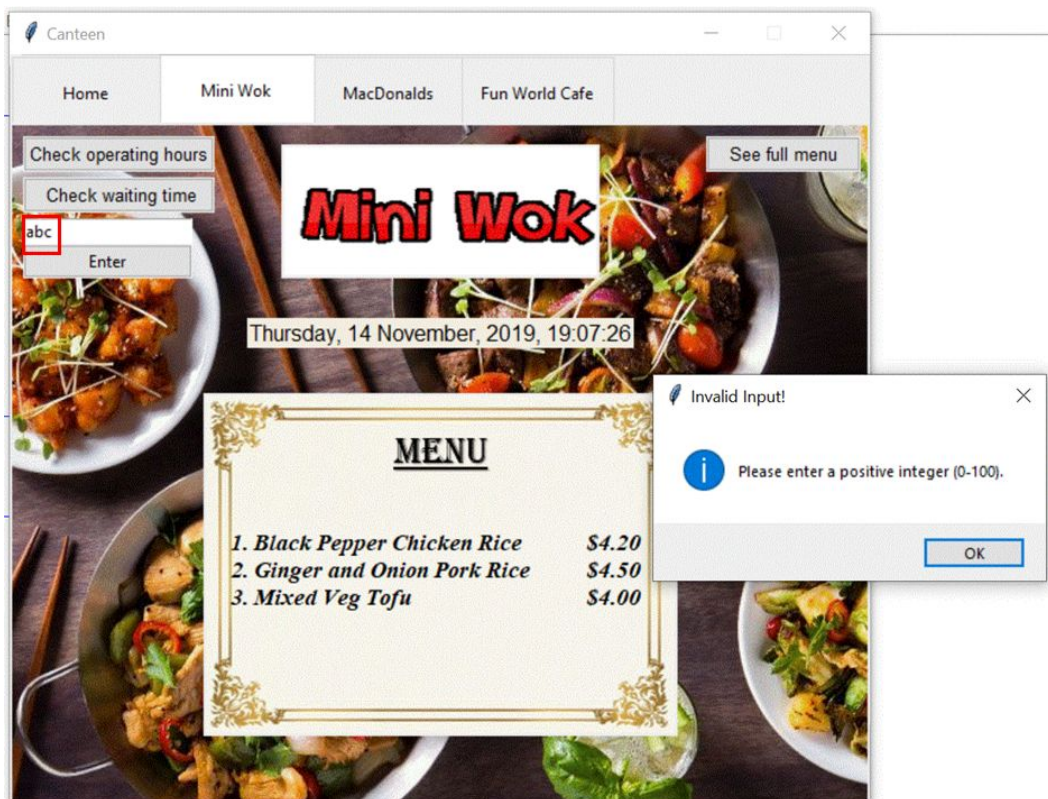
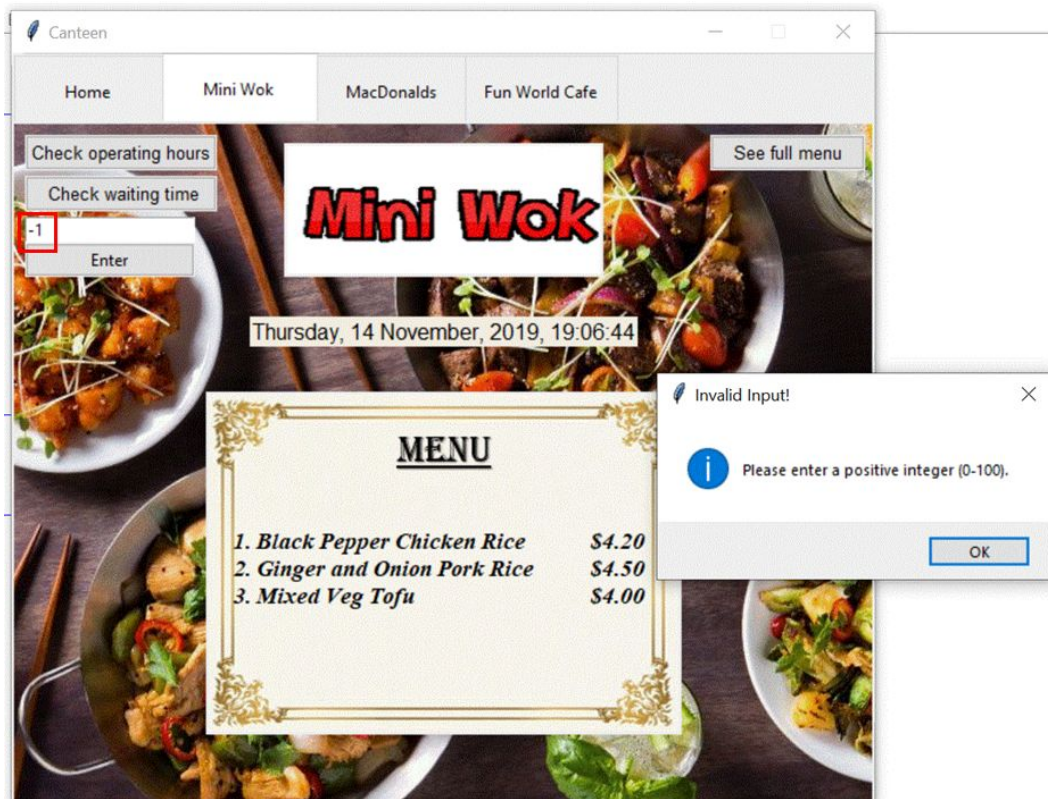




2.2.2 Waiting time entry box error handling

```
def waiting_time_msgbox():
    stores={1:"Mini Wok",2:"Soup Delight",3:"McDonald's",4:"UmiSushi",5:"Fun World Cafe"}
    index=tab_control.index(tab_control.select())
    try:
        ppl = int(no_of_ppl.get())
        if 0<=ppl<=100:
            msg = waiting_time(stores[index],ppl)
            messagebox.showinfo(title= stores[index]+" Waiting Time", message = msg)
        else:
            messagebox.showinfo(title="Invalid Input!", message = "Please enter a positive integer (0-100).")
    except ValueError:
        messagebox.showinfo(title="Invalid Input!", message = "Please enter a positive integer (0-100).")
```

In the `waiting_time_msgbox()` function, the user input for the number of people has to be an integer. The try statement will receive a user input for the number of people in queue. If user input invalid inputs, such as letters and symbols, instead of integers, the except `ValueError` statement will run and an error message will be displayed. Within the try statement, there is an if/else statement and so if number not within 0 to 100, another error message will be displayed.



Chapter 3 Reflection (Nicholas)

3.1 Difficulties encountered

Our team encountered various issues before embarking on the project. This included choosing the preferred Graphical User Interface (GUI) to work on (Tkinter or PyQt5), deciding on the program layout (Notebook, Scrollbar, Combobox), deciding on the ideal file type to store information, splitting the work among the three of us.

During the process of coding, we had problems such as: selecting appropriate widgets (labels, buttons) to display text or call functions, learning callback methods for each widget, learning to pass arguments across functions in a GUI, learning syntax of Tkinter, linking widgets together to simplify code, handling user input from widgets, rectifying program crashes from improper syntax.

3.2 Ways to conquer

Our approach to managing the above-mentioned difficulties are through the philosophy “easier to seek forgiveness than permission”. Code first and rectify later usually yields a shorter code compared to “look before you leap”. Whenever we are uncertain on the syntax of a new line of code, we test the specific line for errors before proceeding.

In the case that one method does not work, we tried other possible methods that may perform the same function. In this aspect, we should be open, creative and persistent. We strived to keep the overall code organised and each function simple. We practiced teamwork through distributing our workload evenly. Finally, it is crucial to learn from resolved problems and expert guidance through avenues such as YouTube videos and helpful websites (stackoverflow.com, tutorialspoint.com).

3.3 Knowledge learnt from this course

This course taught us the syntax for Tkinter which is closely-similar to other Python GUIs such as PyQt5. We learnt that classes hold all objects in a window. Additionally, we have grasped how dictionaries assist in reusability of code through mapping keys to values. We realized how passing an argument “self” while creating widgets makes it available across tabs. Also, we understood that lambda is a crucial keyword that enabled the passing of arguments to functions.

Next, we realized how reusable and clearly-defined functions can enhance readability. Overall, we understood the general structure and elements in a Tkinter GUI (class, Canvas, Frame, widget, callback). Therefore, this course has equipped us with the confidence and knowledge to produce a Tkinter GUI with other applications.

3.4 Further improvement and suggestions

We propose additional guidance for ways to split the workload in each specific platform. This helps each individual to understand their role in the project and perform better. Subsequently, the school could set one day (such as Saturday) for a mandatory coding competition for all students. This enables students to appreciate what they have learnt beyond the textbook. Finally, we believe the school could initiate compulsory enrolment for students in a Python-workshop (such as artificial intelligence). This can provide exposure and further elevate interest in programming.

(1198 words)

Chapter 4

References

<https://stackoverflow.com/questions/7546050/switch-between-two-frames-in-tkinter>

<https://stackoverflow.com/questions/27820178/how-to-add-placeholder-to-an-entry-in-tkinter>

<https://docs.python.org/3/library/datetime.html>

<https://www.programcreek.com/python/example/96412/tkinter.scrolledtext.ScrolledText>

<https://pythonspot.com/tk-message-box/>

<https://docs.python.org/3/library/tkinter.ttk.html>

<https://stackoverflow.com/questions/50625818/how-to-get-the-selected-date-for-date-entry-in-tkcalendar-python>

<https://stackoverflow.com/questions/55012387/expanding-dateentry-display-from-m-d-yy-to-mm-dd-yyyy>

<http://effbot.org/tkinterbook/toplevel.htm>