CZ1003 Mini Project

Real Time Canteen Information System

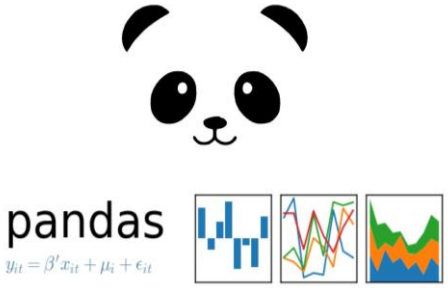
Chapter 1: Introduction

Who are we? What do we hope to achieve?

1.1 Who are we?

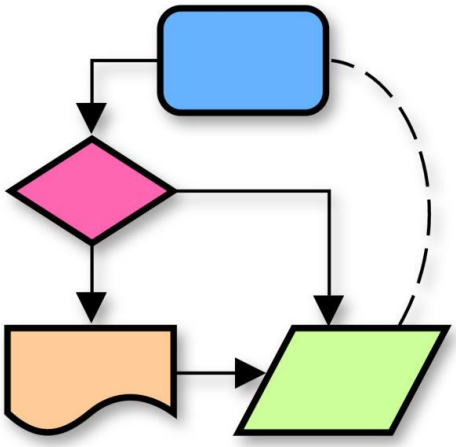
A team, consisting of Jeremy, Joshua and Ernest, designed a real-time NTU North Spine Canteen information system application.

The program was written in **Python** with **Tkinter** and **Pandas**.



Chapter 2: Program Design

Algorithm Design, Program Run-through, File Handling



2.1 Algorithm Design

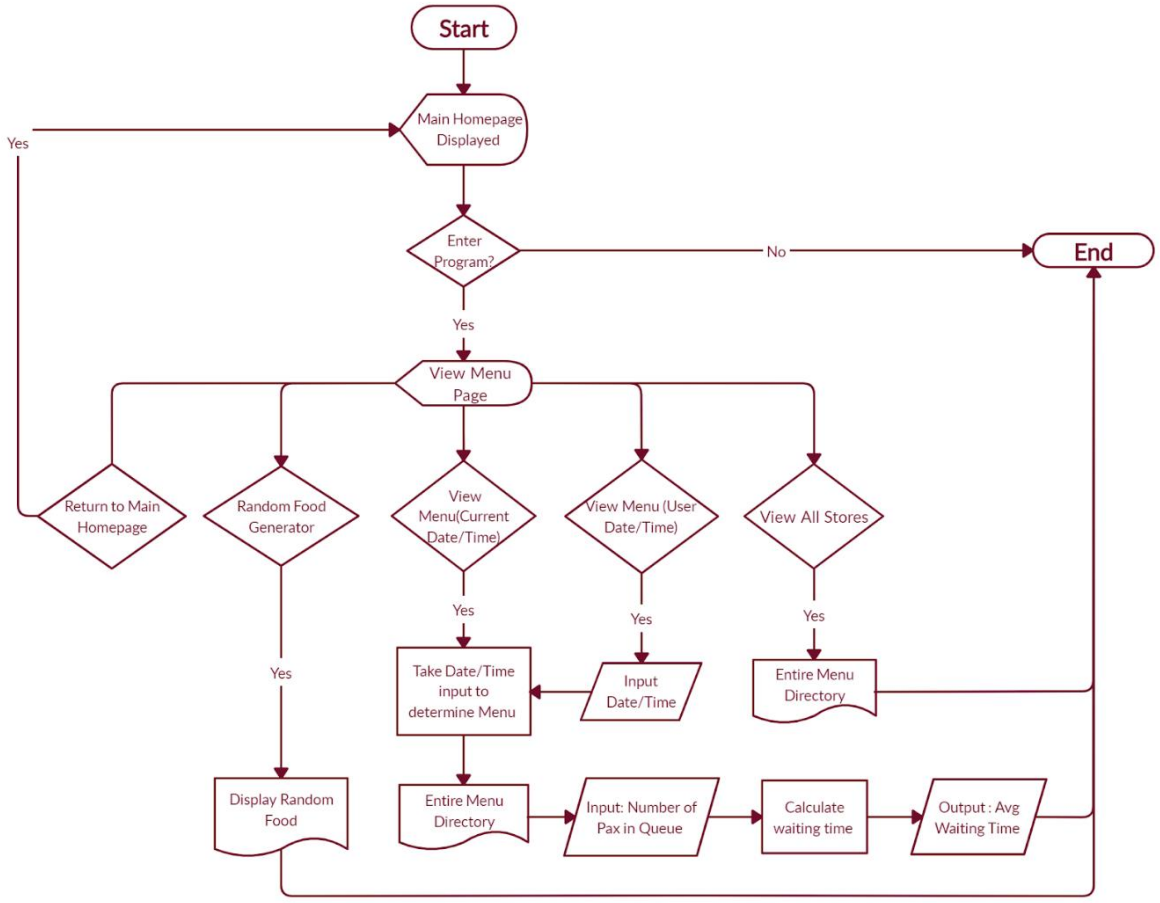


Fig 1.1 Flowchart

Users can view available food options based on either current system's date/time or user-specified date/time. Essential canteen information such as price and operating hours are also displayed.

2.2 Data Storage

	A	B	C	D	E	F
1	Store Name	Operating Hours	Food Item	Price ($)	Availability (Time)	Availability (Day)
2	Yong Tau Foo	0800 - 2000	Wanton Mee	$3.00	AM & PM	Everyday
3	Yong Tau Foo	0800 - 2000	Dumpling Mee	$3.80	AM & PM	Monday
4	Yong Tau Foo	0800 - 2000	Bar Chor Mee	$1.00	AM & PM	Wednesday
5	Yong Tau Foo	0800 - 2000	Tomyam Noodle	$2.50	AM & PM	Everyday
6	Yong Tau Foo	0800 - 2000	Laksa Noodle	$2.80	AM & PM	Everyday
7	Chicken Rice	0800 - 2000	Chicken Rice	$2.50	AM & PM	Everyday
8	Chicken Rice	0800 - 2000	Roast Chicken Rice	$2.50	AM & PM	Everyday
9	Chicken Rice	0800 - 2000	Char Siew Rice	$3.00	PM	Everyday
10	Chicken Rice	0800 - 2000	Roasted Meat Rice	$3.00	PM	Everyday
11	Western Food	0800 - 2000	Breakfast set 1	$3.50	AM	Everyday
12	Western Food	0800 - 2000	Breakfast set 2	$3.50	AM	Friday
13	Western Food	0800 - 2000	Chicken Chop	$5.00	PM	Everyday
14	Western Food	0800 - 2000	Fish and Chips	$5.00	PM	Everyday
15	Western Food	0800 - 2000	Steak	$7.00	PM	Monday

Fig 1.2 Excel file

```
# Program's CSV File
# Change the directory of the file
df = pd.read_csv('/Users/ernestang98/Desktop/Whole_Menu.csv')


# Menu Sorting Function
def menu_display_final(store_name=df['Store Name'], day=df['Availability (Day)'], time=df['Availability (Time)']):
    menu_display = df[
        (df['Store Name'] == store_name) & ((df['Availability (Day)'] == day) | (df['Availability (Day)'] ==
                                                                                 'Everyday')) & (
                (df['Availability (Time)'] == time) | (df['Availability (Time)'] == 'AM & PM'))]
    menu_display = menu_display[['Store Name', 'Food Item', 'Price ($)']].drop_duplicates()
    return menu_display
```

Fig 1.3 Code for storing/sorting the Dataframe

	Store Name	Operating Hours	Food Item	Price ($)	Availability (Time)	Availability (Day)
0	Yong Tau Foo	0800 - 2000	Wanton Mee	$3.00	AM & PM	Everyday
1	Yong Tau Foo	0800 - 2000	Dumpling Mee	$3.80	AM & PM	Monday
2	Yong Tau Foo	0800 - 2000	Bar Chor Mee	$1.00	AM & PM	Wednesday
3	Yong Tau Foo	0800 - 2000	Tomyam Noodle	$2.50	AM & PM	Everyday
4	Yong Tau Foo	0800 - 2000	Laksa Noodle	$2.80	AM & PM	Everyday
5	Chicken Rice	0800 - 2000	Chicken Rice	$2.50	AM & PM	Everyday
6	Chicken Rice	0800 - 2000	Roast Chicken Rice	$2.50	AM & PM	Everyday
7	Chicken Rice	0800 - 2000	Char Siew Rice	$3.00	PM	Everyday
8	Chicken Rice	0800 - 2000	Roasted Meat Rice	$3.00	PM	Everyday
9	Western Food	0800 - 2000	Breakfast set 1	$3.50	AM	Everyday
10	Western Food	0800 - 2000	Breakfast set 2	$3.50	AM	Friday
11	Western Food	0800 - 2000	Chicken Chop	$5.00	PM	Everyday
12	Western Food	0800 - 2000	Fish and Chips	$5.00	PM	Everyday
13	Western Food	0800 - 2000	Steak	$7.00	PM	Monday
14	Western Food	0800 - 2000	Lamb Chop	$7.50	PM	Wednesday
15	Western Food	0800 - 2000	Bangers and Mash	$4.00	PM	Friday

Fig 1.4 Dataframe

Fig 1.5 Canteen information GUI

The canteen information is stored in an excel file (Fig 1.2), converted into a dataframe (Fig 1.4) using the pandas library (Fig 1.3) and displayed to users (Fig 1.5).

2.3 Welcome Page

Fig 1.6 Welcome Page GUI

```
# Program Welcome Page GUI
class WelcomePage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        filler_label = tk.Label(self, text="", bg="Black", font=('Verdana', 1))
        filler_label.pack(fill=tk.X)
        title_label_1 = tk.Label(self, text="Welcome to NS Canteen! Awesome food available here!", bg
                                 font=('Bradley Hand', 25, 'bold'), fg='white')
        title_label_1.pack(fill=tk.X)
        filler_label = tk.Label(self, text="", bg="Black", font=('Verdana', 1))
        filler_label.pack(fill=tk.X)
        opening_hour_label = tk.Label(self, text="Opening Hours: 0800 - 2000", bg="Black",
                                      font=('Verdana', 12), fg='White')
        opening_hour_label.pack(fill=tk.X)
        opening_day_label = tk.Label(self, text="Monday - Friday", bg="Black",
                                     font=('Verdana', 12), fg='White')
        opening_day_label.pack(fill=tk.X)
        filler_label = tk.Label(self, text="", bg="Black", font=('Verdana', 1))
        filler_label.pack(fill=tk.X)
        current_dt_label = tk.Label(self, text="Current Date & Time:", bg="Black",
                                    font=('Verdana', 12), fg='White')
        current_dt_label.pack(fill=tk.X)
        system_date(Frame=self, BG='Black', FG='White', Font=("helvetica", 12))
        system_clock(Frame=self, BG='Black', FG='White', Font=("helvetica", 12))
        bg_label(Frame=self, File=r"C:\Users\Joshua\Desktop\School\Intro to computing\NS canteen info
        picture_button(Frame=self, File=r"C:\Users\Joshua\Desktop\School\Intro to computing\NS cantee
```

Fig 1.7 Welcome Page Code

Beginning with a welcome page, the canteen's operating hours and the current system's date and time are displayed. Users can access the View Menu page or exit the program.

2.4 View Menu Page

Fig 1.8 View Menu Page GUI

```
# Program Date and Time Page GUI
class DtPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        filter_label = tk.Label(self, text="", bg="Black", font=('Verdana', 1))
        filter_label.pack(fill=tk.X)
        title_label_2 = tk.Label(self, text="View Today's Menu", bg="Black",
                                 font=('Bradley Hand', 25, 'bold'), fg='White')
        title_label_2.pack(fill=tk.X)
        filter_label = tk.Label(self, text="", bg="Black", font=('Verdana', 1))
        filter_label.pack(fill=tk.X)
        opening_hour_label = tk.Label(self, text="Opening Hours: 0800    2000", bg="Bl
                                      font=('Verdana', 12), fg='White')
        opening_hour_label.pack(fill=tk.X)
        opening_day_label = tk.Label(self, text="Monday - Friday", bg="Black",
                                     font=('Verdana', 12), fg='White')
        opening_day_label.pack(fill=tk.X)
        filter_label = tk.Label(self, text="", bg="Black", font=('Verdana', 1))
        filter_label.pack(fill=tk.X)
        current_dt_label = tk.Label(self, text="Current Date & Time:", bg="Black",
                                    font=('Verdana', 12), fg='White')
        current_dt_label.pack(fill=tk.X)
        system_date(Frame=self, BG='Black', FG='White', Font=("helvetica", 12))
        system_clock(Frame=self, BG='Black', FG='White', Font="helvetica", 12))
        bg_label(Frame=self, File=r"C:\Users\Joshua\Desktop\School\Intro to computing
```

Fig 1.9 View Menu Page code

Users can either return to the welcome page or view the menu based on :

1. Current system's date and time
2. User-defined date and time
3. View entire menu

2.4.1 View Menu - "Use current date and time"

The menu sorting function (Fig 1.3) filters the dataframe based on current date/time. A new window will display the food options available and the store with the shortest queue.

Fig 2.0 Sorted Menu GUI (i)

Fig 2.1 Sorted Menu GUI (ii)

Upon clicking the store icon, the store's menu will be displayed along with a function to calculate the average waiting time specific to that stall since different stalls have different waiting time per person.

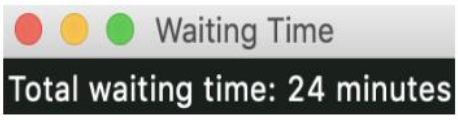


Fig 2.2 Waiting Time GUI

```
def calculate(self):
    calculate_window = tk.Toplevel(self)
    calculate_window.title('Waiting Time')
    calculate_window.configure(background='#192210')
    calculate_window.after(3000, lambda: calculate_window.destroy())
    check_if_digit = (enter_w_time.get()).isdigit()

    if str(Store1) == 'Japanese':
        val = 2
    elif str(Store1) == 'Chicken Rice':
        val = 2
    elif str(Store1) == 'Mini Wok':
        val = 4
    elif str(Store1) == 'Indian Cuisine':
        val = 4
    elif str(Store1) == 'Western Food':
        val = 5
    else:
        val = 3

    def execute():
        if check_if_digit:
            number_of_ppl = int(enter_w_time.get())
            total_wt = number_of_ppl * val
            if total_wt > 60:
                total_wt_label = tk.Label(calculate_window,
                                          text='Total waiting time: At least an hour',
                                          fg="white", bg='#192210')
                total_wt_label.pack(side=tk.TOP)
```

Fig 2.3 Waiting Time Function GUI

Possible scenarios:

1. Valid input
    - Total waiting time (minutes) = number of people in the queue (user input) * waiting time per person for that store (Fig 2.2).
2. Invalid input (non-integer, no input)
    - If '.isdigit()' is false, return error message.
3. Illogical input (100000 number of people in queue)
    - Return 'Waiting time at least an hour' when total waiting time exceeds 60

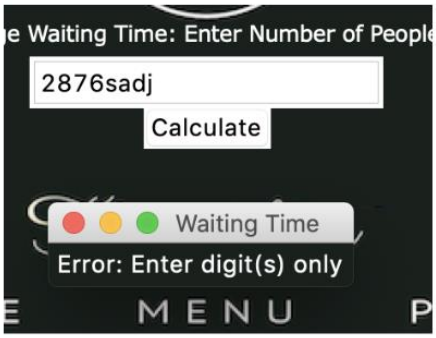


Fig 2.4 Scenario 1

```
def food_generator1():
    current_day = strftime('%A')
    hr = strftime('%H')
    mm = strftime('%M')
    if current_day == 'Saturday' or current_day == 'Sunday':
        closed_frame = tk.Toplevel(self)
        closed_frame.geometry('475x600')
        closed_frame.resizable(width=False, height=False)
        closed_frame.title('Random Food Generator')
        bg_label(Frame=closed_frame, File='/Users/ernestang98/Desktop/bgmenu4.png')
        picture_button(Frame=closed_frame,
                       File="/Users/ernestang98/Desktop/Exit_Button_2.png", BD=0,
                       HT=0, Relief=tk.RAISED,
                       Command=lambda: closed_frame.destroy(),
                       RELX=0.36, RELY=0.9, RELW=0.28, RELH=0.09)
    else:
        if int(hr) < 8 or int(hr) > 20:
            closed_frame = tk.Toplevel(self)
            closed_frame.geometry('475x600')
            closed_frame.resizable(width=False, height=False)
            closed_frame.title('Random Food Generator')
            bg_label(Frame=closed_frame, File='/Users/ernestang98/Desktop/bgmenu4.png')
            picture_button(Frame=closed_frame,
                           File="/Users/ernestang98/Desktop/Exit_Button_2.png", BD=0,
                           HT=0, Relief=tk.RAISED,
                           Command=lambda: closed_frame.destroy(),
                           RELX=0.36, RELY=0.9, RELW=0.28, RELH=0.09)
        elif int(hr) == 20:
            if int(mm) > 00:
```

Fig 2.4.1 Scenario/Error Handling for user input

2.4.2 View Menu - “Use personalised date and time”

A new window opens, requesting user input for date and time.

Possible scenarios:

1. No input
    - Return ‘No input detected’
2. Only time input
    - Return ‘Date input missing’
3. Only date input
    - Return ‘Time input missing’
4. Invalid date/time input
    a. User inputs date in the wrong format
    b. User inputs time in the wrong format
    c. User input date and time in the wrong format
        - Return ‘improper format, re-enter date and/or time’ for 4a - 4c
5. Valid input
    - Returns specific menu, details seen in 2.4.1

```
try:
    if not date_entry and not time_entry:
        no_input = tk.Label(sorry_input, text='No Input Detected', fg='white', bg='#19221D')
        no_input.place(x=225, y=75, anchor='center')
    elif not date_entry:
        no_date = tk.Label(sorry_input, text='Date Input Missing', fg='white', bg='#19221D')
        no_date.place(x=225, y=75, anchor='center')
    elif not time_entry:
        no_time = tk.Label(sorry_input, text='Time Input Missing', fg='white', bg='#19221D')
        no_time.place(x=225, y=75, anchor='center')
    else:
        year, month, day = map(int, date_entry.split('-'))
        my_date = dt.date(year, month, day)
        my_day = my_date.strftime("%A")
        date_input = my_day
        hour, minute = map(int, time_entry.split(':'))
        if date_input == 'Saturday' or date_input == 'Sunday':
```

Fig 2.5.1 Scenario/Error Handling for user input

2.4.3 View Entire Menu

STORE	HOUR	ITEM	PRICE	AM/PM	DAY
Yong Tau Foo	0800 - 2000	Wanton Mee	$3.00	AM & PM	Everyday
Yong Tau Foo	0800 - 2000	Dumpling Mee	$3.80	AM & PM	Monday
Yong Tau Foo	0800 - 2000	Bar Chor Mee	$1.00	AM & PM	Wednesday
Yong Tau Foo	0800 - 2000	Tomyam Noodle	$2.50	AM & PM	Everyday
Yong Tau Foo	0800 - 2000	Laksa Noodle	$2.80	AM & PM	Everyday
Chicken Rice	0800 - 2000	Chicken Rice	$2.50	AM & PM	Everyday
Chicken Rice	0800 - 2000	Roast Chicken Rice	$2.50	AM & PM	Everyday
Chicken Rice	0800 - 2000	Char Siew Rice	$3.00	PM	Everyday
Chicken Rice	0800 - 2000	Roasted Meat Rice	$3.00	PM	Everyday
Western Food	0800 - 2000	Breakfast set 1	$3.50	AM	Everyday
Western Food	0800 - 2000	Breakfast set 2	$3.50	AM	Friday
Western Food	0800 - 2000	Chicken Chop	$5.00	PM	Everyday
Western Food	0800 - 2000	Fish and Chips	$5.00	PM	Everyday
Western Food	0800 - 2000	Steak	$7.00	PM	Monday
Western Food	0800 - 2000	Lamb Chop	$7.50	PM	Wednesday
Western Food	0800 - 2000	Bangers and Mash	$4.00	PM	Friday
Western Food	0800 - 2000	Bolognese Spaghetti	$5.00	PM	Everyday
Mini Wok	0800 - 2000	Fried Rice	$3.00	AM & PM	Everyday
Mini Wok	0800 - 2000	Sambal Fried Rice	$3.50	PM	Everyday
Mini Wok	0800 - 2000	Chilli Beef Rice	$5.00	PM	Everyday
Mini Wok	0800 - 2000	Seafood Horfun	$5.00	PM	Friday
Mini Wok	0800 - 2000	Beef Horfun	$5.00	PM	Thursday
Japanese	0800 - 2000	Omu Rice	$3.50	AM & PM	Everyday
Japanese	0800 - 2000	Saba Fish Set	$3.50	AM & PM	Everyday
Japanese	0800 - 2000	Sashimi Platter	$3.00	PM	Everyday
Japanese	0800 - 2000	Sushi Platter	$3.00	PM	Tuesday
Japanese	0800 - 2000	Tempura Set	$3.00	PM	Thursday
Indian Cuisine	0800 - 2000	Samosa	$0.70	AM & PM	Wednesday
Indian Cuisine	0800 - 2000	Chicken Masala	$3.00	PM	Everyday
Indian Cuisine	0800 - 2000	Plain Prata	$0.70	AM & PM	Everyday
Indian Cuisine	0800 - 2000	Egg Prata	$1.10	AM & PM	Everyday
Indian Cuisine	0800 - 2000	Cheese Prata	$1.50	AM & PM	Everyday
Indian Cuisine	0800 - 2000	Mutton Masala	$3.00	PM	Tuesday

Fig 2.6 Full Menu GUI

Each column of the dataframe (Fig 2.6) is converted into a string label. These labels are then packed and displayed in a window for users to view.

2.5.1 Additional Function 1 - Random Food Generator (RFG)

```
# Random Food Generator Function
def random_food(day1, time1, num1, frame1=None):
    menu1 = (menu_display_final(day=day1, time=time1))
    rand_fd = menu1.sample(n=int(num1), replace=True)

    def structured_menu(frame1):
        store_name_list = rand_fd['Store Name'].to_string(index=False)
        food_item_list = rand_fd['Food Item'].to_string(index=False)
        price_list = rand_fd['Price ($)'].to_string(index=False)
        label1 = tk.Label(frame1, text=store_name_list, fg="white", bg='#000000')
        label1.config(font=("Verdana", 14))
        label1.place(x=90, y=360, anchor='center')
        label2 = tk.Label(frame1, text=food_item_list, fg="white", bg='#000000')
        label2.config(font=("Verdana", 14))
        label2.place(x=240, y=360, anchor='center')
        label3 = tk.Label(frame1, text=price_list, fg="white", bg='#000000')
        label3.config(font=("Verdana", 14))
        label3.place(x=400, y=360, anchor='center')

    structured_menu(frame1)
    # return generated random food
    return rand_fd
```

Fig 2.7 RFG Code

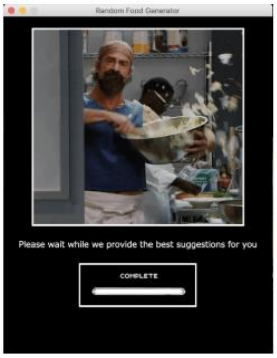


Fig 2.8 RFG GUI

This function filters food options available based on current date/time and requests user for the number of customers to generate random food options for. Taking the user input as a parameter, the function uses *"DataFrame.sample()"* method to return a random sample of user-specified size. This information is displayed in the GUI as seen in Fig 2.

Possible scenarios:

1. Valid input
    - Return generated food options
2. Invalid input (non-integer, no input)
    - If '.isdigit()' is false, returns 'Error. Enter Digits(s) only'
3. Illogical input (more than 20)
    - Returns 'Error. Enter up to 20 people per input'
4. Date/Time error - Outside of operating hours
    - Returns 'Food court is closed'

2.5.2 Additional Features 2 - Shortest queue

This function filters the stores based on date/time selected and displays the store with the shortest queue based on that. Stores are given probabilities of having the shortest queue, based on the popularity of the stores at different timings. Taking the specific date/time as parameters and using *'random.choices()'* method, the function returns that store based on the respective probability.

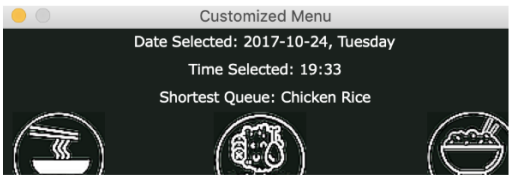


Fig 3.0 Shortest Queue Function GUI

```
def shortest_queue():
    while True:
        store = ['Yong Tau Foo', 'Chicken Rice', 'Western Food', 'Mini Wok',
                 'Duck Rice', 'Indian']
        hour_input = hour
        prob_choice = queue_prob(hour_input)
        if prob_choice == 0:
            break
        else:
            shortest_queue = random.choices(store, prob_choice, k=1)
            return shortest_queue
            break
```

Fig 3.1 Shortest Queue Function Code

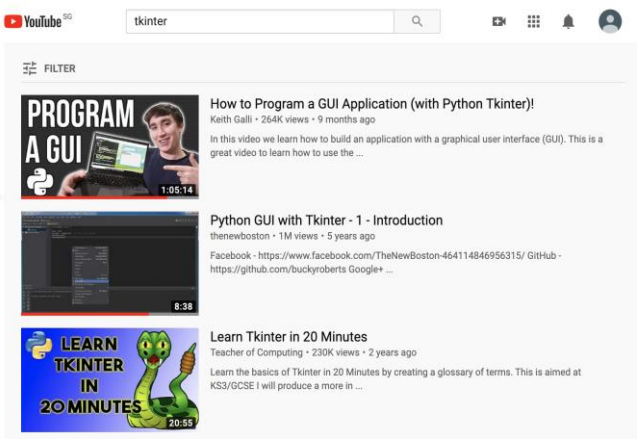
2.5.3 Additional Features 3 - Self-destruct windows

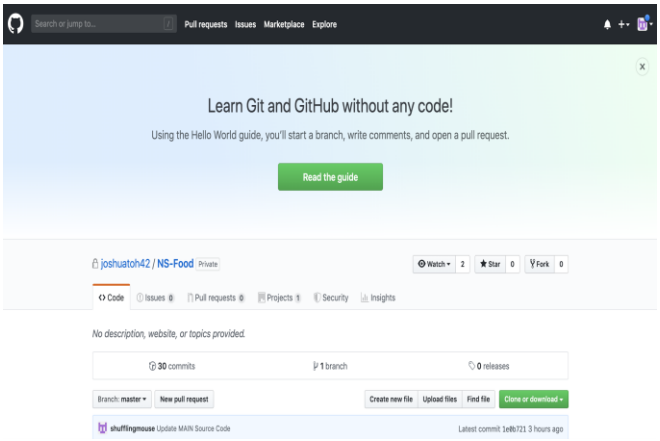
Invalid input and waiting time windows will automatically close after 3 seconds.

Chapter 3: Reflection

Difficulties encountered, learning points

3.1 Learning Tkinter





Being our first time coding a graphic user interface (GUI), we struggled. This led us on a path of self-learning, spending countless hours on YouTube tutorials and various coding forums. To smoothen this process, we created a private project on GitHub which facilitates group learning, cooperation and contribution. We were able to cover each other blindspots while learning and coding the GUI.

3.2 Project approach

After reading the project brief, we learnt Tkinter individually. We got overly enthusiastic trying to apply what we learn and started on the GUI immediately. Although we were able to explore and built a myriad of Tkinter functions, we lacked a concrete plan which only slows our progress of building the program. Subsequently, we first mapped out the necessary details such as flowcharts and required functions. This resulted in us staying focused and clear of the task at hand. To keep everyone updated, we also held weekly meetings which prove critical to the project.

3.3 Splitting the workload

We segmented the project into 2 parts: the front-end GUI and back-end code. Our plan was to finish them individually and combine them into one massive source code. Initially, integration was easy as the components were relatively small. As our code got larger, combining codes became problematic as our syntaxes are different. For example, Joshua's GUI for the sorted menu was done through 'Pandastable' library. However, Ernest's code requires the code to be in a string format and the function had to be converted into other formats. Moving forward, communication is essential to avoid building incompatible codes.

3.4 Using Widgets

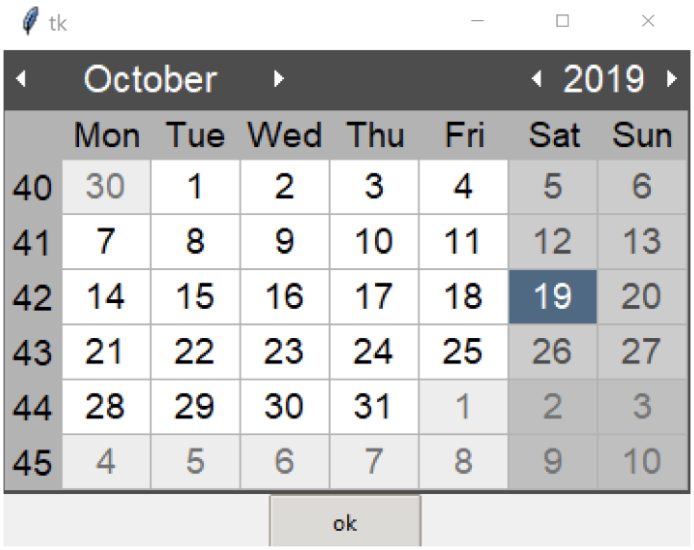


Fig 3.1 Calendar Tkinter Widget

When requesting for user inputs, providing options using Tkinter's widgets instead of open-ended responses from users might prove to be successful in reducing input errors. For instance, the 'Calendar' widget may be used to request for date input. This reduces date input errors and hence, reduces the complexity of file and error handling as the user cannot deviate from the choices that are presented by the program.

3.5 Additional Functions' Accuracy

For the shortest queue generator, the information we coded was based on our experiences as patrons of the North Spine Canteen. We could have further improved the accuracy of the function by gathering factual or even real-time data on the average number of people queuing for a particular store at a particular time, and on a particular day

3.6 Project Contribution

Name	Contribution
Ernest	Chapter 1.1 — 2.3 & 3.1 Create GUI in Tkinter Create user input Create welcome page Create visuals for menu
Joshua	Chapter 2.4 — 2.4.3 & 3.2 — 3.3 Create pandas function Create date time page Create visuals for buttons Error handling
Jeremy	Chapter 2.5.1 — 2.5.3 & 3.4 — 3.5 Create clock function Create view menu page Create visuals for Home page Create calculate waiting time function Create additional functions Create GIF

Appendix:

Report's Image Source

Page 1:

1. Cafeteria Clipart, retrieved from https://www.pngrepo.com/svg/132681/cafe

Page 3:

1. People Clipart, retrieved from https://icon-library.net/icon/who-we-are-icon-28.html

Page 4:

1. Python Logo, retrieved from https://commons.wikimedia.org/wiki/File:Python-logo-Notext.svg
2. Pandas Logo, retrieved from https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSQTysT6IrVEe35oHNnm5zckg6S5wU4oWt4VYnxOEU2rNBKx7c_og&s

Page 5:

1. Flowchart Clipart, retrieved from https://icon-library.net/icon/flow-diagram-icon-4.html

Page 13:

1. Thinking Clipart, retrieved from https://thenounproject.com/term/self-reflection/114599/