Program assessment rubrics

Lab Group: DD2

Names of members: Adithya Venkatadri Hulagadri
Balamurugan Darshini
Callista Rossary Chang

Include this page in your softcopy submission. You are required to fill in the first 4 entries in the table, indicating how you have made use of these CT concepts in your software design.

Assessment Criteria		Description/Comment	Marks
1	Use of Pattern recognition (10)	**1. Storing and processing NTU red and blue bus loops. [Transport.py]** → Bus stops are ordered so we developed an algorithm to read lists in order and allow us to store loops in lists by using a modulus operator **2. Tuples within dictionary. [Dictionary.py]** → We used nested lists and tuples inside a dictionary to compress data in lieu of having multiple separate dictionaries. **3. Autofill feature. [inputscreen.py]** → Using the pickle module and multiple for-loops, we recommend to the user food preference, min budget and max budget according to the user's past 10 choices.	
2	Use of Abstraction (10)	**1. Use of dictionaries to hold data for canteens/menus. [Dictionary.py]** → We represented canteens' stalls by reducing them to a few variables: food tags, price ranges and location co-ordinates. **2. Use of dictionaries to hold data for bus routes/coordinates. [Transport.py]** → We represented the bus stops by reducing them to their coordinates. We modelled the continuous path by approximating it using straight-line segments by introducing waypoints. **3. Calculating distance. [Backend.py]** → To represent the problem of finding the nearest canteen, we took the approach of finding the shortest distance by approximating the real life walking distances with straight line distances on the map for easy computation.	
3	Use of decomposition (10)	**1. Use of a master console decompose the program parts. [MasterConsole.py]** → We decomposed our program into 5 different parts and integrated it all together into one main console. Master console coordinated the different screens and transferred information from one to another. It imports functions on a need-to-know basis. Flow of parts, in order: 1. *inputscreen.py:* User GUI to get user's location, food preference, min and max budget inputs; displays user's past inputs, has auto-	

		input/recommendation feature 2. *Dictionary.py & Backend.py*: Access food database dictionary, search and sort through to return results according to user's inputs 3. *outputscreen.py*: Then, results are shown on the screen, where buttons can be pressed to show the transport route. 4. *Transport.py*: Search through dictionary of bus stops/coordinates to return coordinates, returning route coordinates based on user's coordinates and coordinates of suggested canteen. 5. *pathdisplay.py*: Then, the coordinates of the bus stops needed to get to the suggested canteen is graphically displayed onto the screen. If the distance is close enough the display will suggest the user to walk.	
4	Algorithm Design (20)	**1. Backend searching operation. [Backend.py]** → We used a linear search algorithm in order to search the dictionary by user's food preference, min budget and max budget inputs, returning a dictionary of a reduced size after filtering. **2. Backend sorting operation. [Backend.py]** → We used a bubble sort algorithm in order to sort the reduced dictionary of canteens by distance, then showing the top 5 results to the user from closest distance to furthest distance. **3. Design and features of GUI. [inputscreen.py, outputscreen.py, pathdisplay.py]** → In our input screen, we took user's keyboard input directly from the GUI instead of the console. This is to get user's location, food preference, and minimum and maximum budget. To do this, we took note of any of the user's alphanumerical key-press events as well as handle special keys e.g. hyphen keys. If the user presses backspace, the list splicing method is used to delete the last character. The user indicates his location by clicking on the map, in which a pointer icon will be displayed, and the coordinates will display at the top left screen. There are also dynamically generated error messages that will be displayed in the case of invalid user inputs. The program works differently depending on whether or not the user chooses to input food preference or budget (only user location is necessary for sorting). We used the pickle module to store user's past 10 inputs into a backup file (savefile.out) and used this information for the autofill/recommendation feature, and also displays the user's input history. In our output screen, we created a smooth interface between input, output and transport display screens, allowing the user to move seamlessly from one screen to another, using dynamically generated buttons. When the user clicks on a specific result, it leads to the transport screen to show them the way there by bus or foot. In our transport screen, the backend first finds the nearest bus stop to the user, then finds out the nearest bus stop to the destination, and then computes the fastest/most efficient bus route and displays the stops on the map. This is an effective visual aid.	

		4. Dealing with invalid inputs and exceptions. [inputscreen.py] We forecasted possible exceptions namely blank inputs, null inputs, invalid inputs, invalid data types and mitigated the risks of invalid data types being accepted into the backend processing.	
5	User Interface Design (10)		
6	System Complexity (10)		
7	Teamwork & Presentation (10)		
8	Individual Oral Assessment (20)		

Others (Optional)

Date of Assessment:_____ By:_____