MA1008 Introduction to Computational Thinking
Mini Project: MRT Information System
Semester 2, AY 2024/2025, Week 10 – Week 13

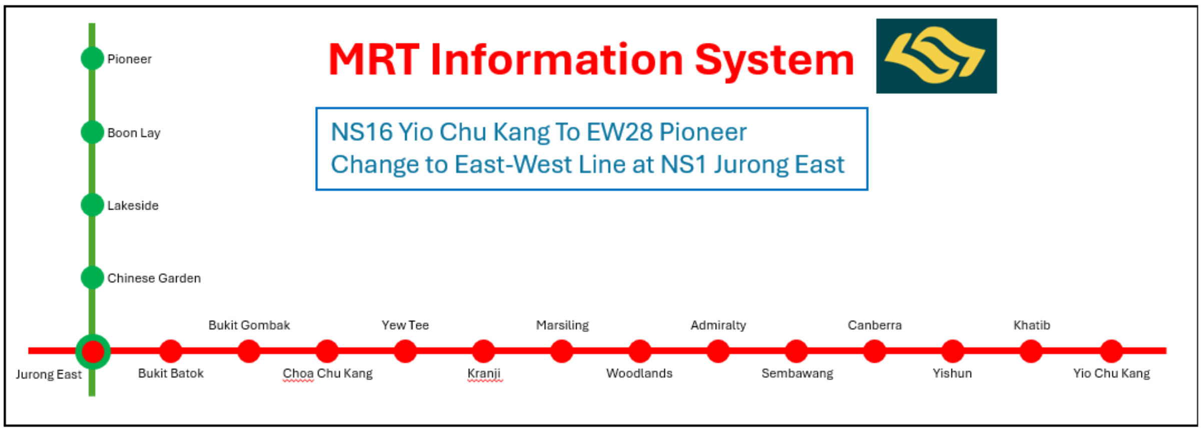
1. Introduction

The objective of the mini project is for you to produce a program of a moderate size and depth that will require you to utilise what you have learned in this course, and a bit more, to do something useful and interesting. Through this, you will learn to design, manage and execute a sizable program.

2. The Project

We are all frequent commuters on MRT trains and often have queries about the train routes, schedules, and so forth. It would be good if there is an app which can answer our queries. Therefore, this project aims to develop a program for displaying information of the Singapore MRT system. A user of the program would make a certain query, and the program would then display the required information nicely (and correctly, of course) in a window on the screen. This screen may be your mobile phone screen or a screen standing outside an MRT station. Therefore, the information displayed must be clear and unambiguous, and the display needs to be attractive.

The program would need to have the data about the MRT lines and their schedule and, given a query, extract the relevant information for display. More on the data shortly. Here is an example of a query and a possible resulting display: Find the best route from Yio Chu Kang to Pioneer.


MRT Information System

NS16 Yio Chu Kang To EW28 Pioneer
Change to East-West Line at NS1 Jurong East

2.1 Your tasks and the MRT data

Your task is to write a program to allow users to pose queries about the Singapore MRT system and then provide the answer, displayed in a graphics window.

The data on the Singapore MRT system is provided in the text file MRT.txt. The file contains the data of every station in the order they appear on an MRT line. One line in the file has eight data items, separated by semi-colons:

(1) Serial number
(2) A station (S1) and the next station (S2) it is connected to
(3) The MRT line of these two stations
(4) The distance between S1 and S2
(5) The time of the first train at S1 going to S2
(6) The time of the last train at S1 going to S2
(7) The time of the first train at S2 going to S1
(8) The time of the last train at S2 going to S1.

In Item (2), a station has two identifiers: the station code, such as EW1, and the station name, such as Pasir Ris. A station may be identified by the code or the name.

Also provided is a Word file, MRT.docx, containing the same data but displayed in a table, so that you can see the data more easily. This is for your reference only and your program need not handle it. This file also contains the data for the LRTs, but these are not required for your program.

Separately, the average speeds of the trains on each line are (to the nearest km/hr):

North-South	41 km/hr
East-West	43 km/hr
Circle	36 km/hr
Thomson-East Coast	40 km/hr
North East	34 km/hr
Downtown	38 km/hr

Note: these speeds are not obtained from an official source, but derived from data available, and should not be construed as the actual. Use them only for the purposes of this project.

2.2 Actions you or your program need to take

Your program needs to read in the data from MRT.txt and store them in a data structure, together with the average speed of the trains on each MRT line. Once that is done, your program needs to use the data to take queries and display the answers in a form like the example above.

There are four queries which your program must handle:

1. Given an MRT line, list all its stations in order.
2. Find and display the best route between two stations. Users are free to choose the stations.
3. Display also the distance and the time of travel between the two stations.
4. Given the start time of a journey at a specific station, state if there is a train running and whether it is would arrive at the destination station before the last train time.

Furthermore, your program also needs to handle at least four more queries of your own choosing. A couple of examples: What MRT line(s) is a given station on? What are the inter-change stations between two given MRT lines?

You may make your system more comprehensive by providing more data of your own beyond those listed in Section 2.1. For example, you may add peak hours and the fare structure as data for the system. Then you may have this query: State the fare from Station A to Station B during peak hours. For the fare structure, visit https://www.ptc.gov.sg/fare-regulation/bus-rail/fare-structure.

2.3 The form of the queries

You are not expected to handle queries expressed freely in English. Instead, you are to provide the eight queries in a list, from which the user is to select. Each query would require further inputs. For example, if the query is whether a passenger would arrive at the destination before the time of its last train, then your program would need to ask for three inputs: the start station, the start time and the destination station.

It would be tedious if the user must type in the name of an MRT line or a station in full. To ease the pain, you may consider accepting the first few letters of a name, so long as it is unique, such as Dh for Dhoby Ghaut, or E for East-West Line. A station may be identified by its code, such as EW1 for Pasir Ris. Note that an interchange station has multiple codes, one for each line it is on, but only one name.

2.4 Program Design

a. A program needs to be designed before you start any coding. Work out first how you would store your data and the form of the dialogue between the program and the user.
b. Most queries should be coded as functions, and your main program should be fairly short, consisting mainly of calls to these functions.

c. Where a set of statements is repeated in different places in the program, consider putting it into a function, and call the function where the repetition occurs.
d. When a repetition occurs but with different variables and outputs, then the variables probably should be parameters to your function, and the outputs should be the returned values.

2.5 Computations

Most of your computation would be in navigating through the data structure, to get to the data you need. For example, to calculate the distance between two stations, you would need to visit the stations in between and extract the distance between two adjacent stations. There aren't any deep mathematical calculations.

2.6 Graphics

The query outcomes are to be displayed in a graphics window, with graphical illustration where helpful. This is to be done using the graphics library called Turtle. Turtle capabilities and functions are described in Section 24.1 of the Python documentation, which you can access via the IDLE interactive shell by clicking Help > Python Docs and then search for "turtle".

In doing the drawing, you need to consider a few basic things.

a. Your display window is fixed, in terms of the number of pixels in the x and y direction. (Turtle gives you a default window size if you do not specify it yourself.) Your display extend is variable, as the number of stations varies depending on the journey. Hence, you should handle the size of the window carefully.
b. Turtle provides you with many functions for doing display, but you only need to use a few of them for the purpose of this project, mainly for posting text, drawing straight lines and dots, and selecting colours. So, determine the graphics functions you need before reading the Turtle documentation. That would save you some time.
c. By default, Turtle draws slowly to allow us to see how the drawing is done. However, for this project, you should draw quickly. So make sure that you set the drawing speed to the fastest possible, again doable by calling the appropriate Turtle function with the appropriate parameter value. You should hide the turtle too.

2.7 Program Flow and Control

a. When running the program, it should be clear at every stage what the user needs to do.
b. Therefore, you need to ensure that the information presented on the screen, such as prompts for user inputs, is clear and unambiguous.
c. When asking the user to make a selection, always require the minimal input from the user to reduce burden and error.
d. Allow the user to select between ending the program and returning to input for a new start.
e. Your program should trap and handle errors appropriately. For example, in the case of an input error, your program should tell the user what the error is and allow the user to re-input.
f. Your program should be easy to read and understand. So, make sure that it is well commented, well modularized and uses meaningful (but not overly long) variable and function names.

3. Prohibitions

Apart from the graphics library, everything you need for this project can be done through what you have learnt in this course. To allow you to exercise what you have learnt and to prevent you from veering wildly beyond the scope of this project, your program should:

a. Not use the *class* construct to define objects you require in your program
b. Only use the *Turtle* library for graphics. You must not use *tkinter, matplotlib, plotly* or similar.
c. Not use the libraries *numpy, pygame* or *json*.
d. Not use the *lambda* construct or the *eval()* function.

You may use the standard Python libraries like *math* and *datetime*. Beyond that, using any external library requires the approval by your tutor. So, please consult your tutor beforehand.

4. Conduct of the Project and the Submission

4.1 Schedule

The duration of this project is from the Monday of Week 10 to the Sunday of Week 13, spanning four weeks. The class hours in Weeks 11-13 are dedicated to the project, apart from CA4 in Week 11 for one hour. Your tutors will be in the class to offer you help. Make full use of them.

The project deadline is at **23:59 hours, Sunday 20 April 2025.**

4.2 Project Log

You are required to keep a project log in a Word or Excel file, listing the things you have done for the project day by day. You are required to submit the file weekly at the end of each of the four weeks, together with the Python program you have written up to that time. This program does not need to be complete or fully working, it merely serves as a record of your progress. This log should be cumulative, which means one week's log should be added to the end of the previous week's log, without changing anything of the previous week. A sample project log is provided separately.

See Section 4.4 on how to submit the log. **The weekly logs and programs show your interim progress in the project and are worth 15% of the project grade.**

In the four project weeks, your tutor may seek to speak to you concerning your progress, your log or your program. So, you should be present during the tutorial hours.

4.3 What you need to submit finally

i. **A working Python program** that gets the relevant inputs from the user and displays the required information in a graphics window.
ii. Your weekly log for Week 4.
iii. **A report in a Word or PDF file**, providing
   - A guide on how to run your program
   - A list of the queries your program handles, including their input requirements.
   - The graphical outputs from your program for the queries you have.
   - Highlights of the **key strengths and limitations** of the program.
   - Highlights of features you consider to be worth some bonus marks, such as if you also include your own extra data and queries.

4.4 How to Submit

Please submit all your files through the course site at the same item where you fetched this project file. The title of the item is a link to the submission page, which has a link for uploading your files. This applies to the submission of the project logs and the interim programs as well as the final submission.

5. Grading Rubrics

Here are what the graders will be looking for when grading:

i. The ease in interacting with the program, which includes specifying the input data and working with the input files.
ii. The quality and correctness of your program, which includes:
   a. How easy it is to read and understand your program. This means your program should be adequately commented with good choice of variable and function names.
   b. Logical program flow.
   c. Modularisation of the program including appropriate use of functions, the design of the functions which include the appropriateness of parameters in the functions.
iii. The quality and correctness of the outputs.
iv. Bonus marks, up to a maximum of 20, will be awarded for the inclusion of extra data of your own and the associated queries. Hence potentially, you can score a maximum of 120 marks. Marks beyond 100 will be added to your CA1 or CA3 marks.

6. Epilogue

You should start working on the program immediately. Do not procrastinate. What you produce depends on the time you spend on the project and your ability in creating good algorithms and writing good code.

One of the major problems of past projects for the tutors is that there is no way to tell how to run a program. The program must offer help to the user directly (using prompts and on-screen instructions) on what input is required at every stage. Quite often, the input is very cumbersome, such as requiring the user to type a long string exactly. You need to help your user by making your input requirements as obvious and as simple as possible. For example, it is good to allow unique abbreviated input for a station name, instead of requiring the full name. Of course, your user must know what abbreviation is allowed.

This is an individual project. You may consult the tutors and discuss with your classmates, but everyone must write their own program. **Any programs deemed to have been copies of each other will be penalised heavily, regardless of who is doing the copying.**

If in doubt or in difficulty, always ask. And ask early!