**London Tube Navigation Expert System in CLIPS**

**PART - 2**

**Team 4**

**Chadha, Harshita**

**Sanghvi, Prima**

**Tadikonda, Naga Sai Bhavya**

Department of Computer Science

George Washington University

**Professor Stephen Kaisler**

Department of Computer Science

George Washington University

Submitted on - **November 11, 2023**

**TABLE OF CONTENTS**

1. Introduction
2. Files Submitted and Instructions for Running the Expert System
3. System Overview, Functionality and Examples
4. Facts, Templates and Data Collection
   1. Data Collection
   2. Revisions in Templates and Facts
   3. Templates in CLIPS
   4. Facts in CLIPS
   5. Loading Facts and Templates in CLIPS
5. Design and Implementation of CLIPS Rules
6. Conclusions and Future Improvements
7. References

**1 - Introduction**

The London Tube is a complex transportation system comprising 11 lines, covering nine zones, and offering connections to various rail services. This report serves to provides an overview of part 2 of the project wherein we aimed to create an expert system that allows travellers to navigate through the complex London underground network.

Our goals in this phase were to explore in detail the requirements of creating such an expert system, to modify the facts and templates created in part 1 of the project to suit these changes, and to create suitable rules and functions to obtain the desired outcome. Further, we also worked on creating a simple, option based interface that allows users to interact with the expert system and use the information contained in the knowledge base to plan their next exciting outcome in London!

This report is made up of seven sections. The first and present section serves as an introduction to the problem and an overview of the work done to address it. The second section provides an overview of the submitted files and steps on how the submitted CLIPS files can be run to interact with the system. The third section describes in detail the system functionality with attached screenshots for demonstration. This is followed by sections four and five that provide information on the rules and functions created in this phase of expert system development as well as the changes made to the templates from phase one. Lastly, conclusions, future direction, and references for all external sources are provided.

**2 - Files Submitted and Instructions for Running the Expert System**

In the file exchange for group 4 on blackboard, 6 items have been submitted. All the files have been named using the convention Part II-Group4-\*file-name\*.\*extension to distinguish from part 1 submissions. The list of submitted files with a brief description of their contents is provided in table 1 below -

**Table 1: Description Of Submitted Files On Blackboard**

| **File Name** | **File Type** | **Description** |
| --- | --- | --- |
| Part II-Group4-Report.docx | Word Document | A detailed report on the London Tube Expert System development and navigation. |
| Part II-Group4-all-facts.clp | CLIPS file | CLIPS executable file to load all facts for the expert system. |
| Part II-Group4-all-rules.clp | CLIPS file | CLIPS executable file to load all rules and functions for the expert system. |
| Part II-Group4-text-form-facts-and-templates.docx | Word Document | A document containing all templates and facts used for system development |
| Part II-Group4-text-form-rules.docx | Word Document | A document containing all the rules used for system development |
| Part II-Group4-text-form-functions.docx | Word Document | A document containing all the functions used for system development |

In order to interact with the created expert system, two CLIPS files have to loaded into the IDE using the load command followed by the reset and run commands. A sample template for this has been provided below -

(load “/\*\*path to the file\*\*/Part II-Group4-all-facts.clp”)

(load “/\*\*path to the file\*\*/Part II-Group4-all-rules.clp”)

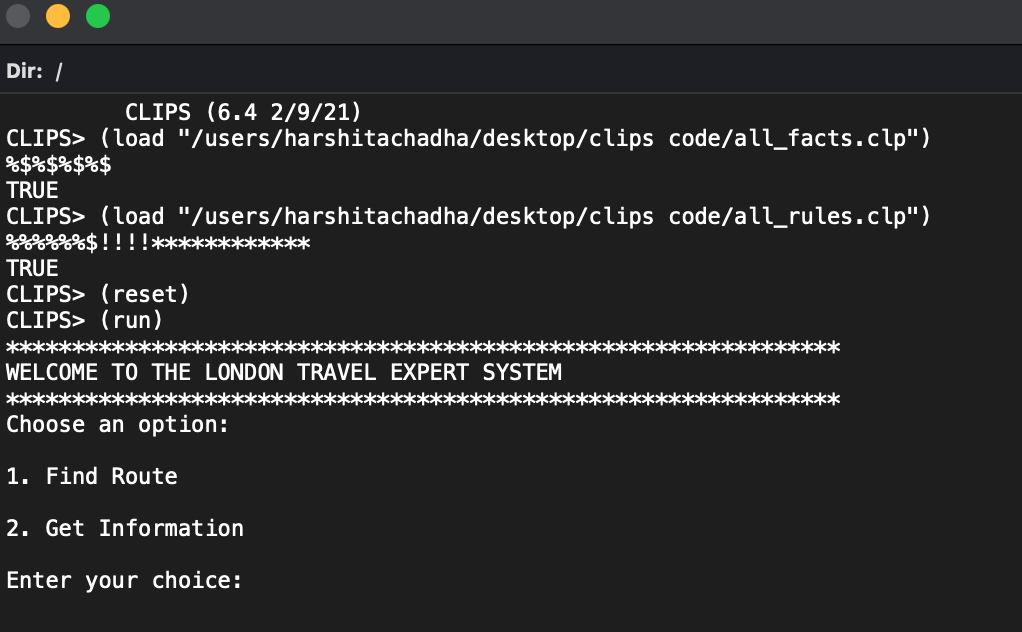
(reset)

(run)

Entering the above commands initializes the system and a menu driven, prompt like expert system is loaded which can be interacted with to plan trips, get more information on attractions in London, etc.

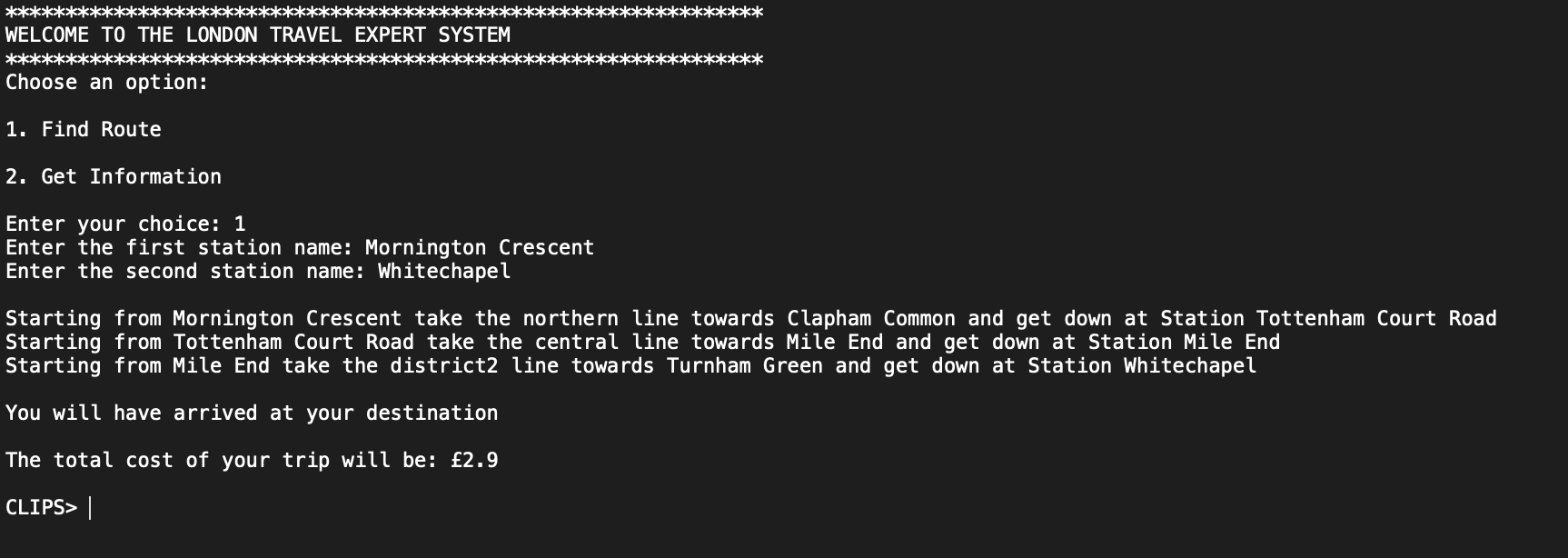
**3 - System Overview, Functionality and Examples**

In this section we provide an overview of the capabilities of our London Tube Navigation System and demonstrate some examples of usage. When the commands illustrated in section 2 above are executed, the system is intialized and ready for interaction. This initial interface is shown in the figure below -



**Figure 1: Initial User Interface For The London Tube Expert System**

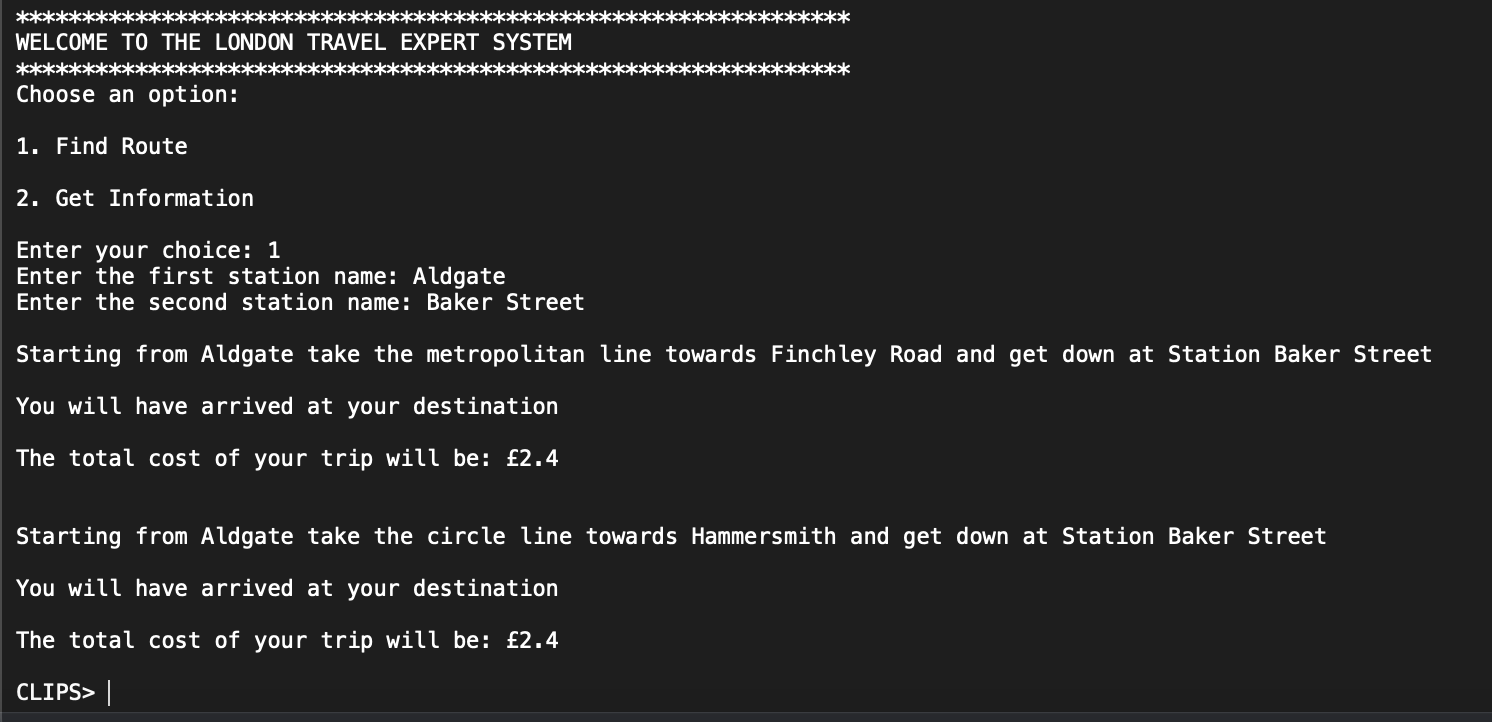
The initial interface offers the user two choices - Find Route and Get Information. When the first option, Find Route, is selected, the user is prompted to enter the name of the start and the ending station. This is shown in the figure below -



**Figure 2: Calculating Route Using The Expert System**

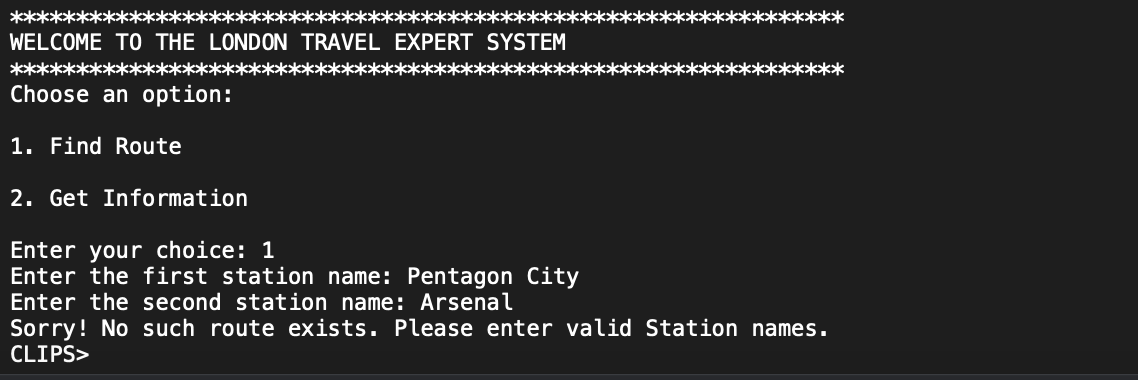
As can be seen, entering the name of the stations generates the route for the trip along with the cost of making the trip. In order to calculate the route, a dynamic fact is initialized to capture the user input. Following this, if the route to be calculated happens to be for stations on the same line, a different rule executes simply outputting the path.

If in case the stations happen to be on different lines, a separate rule executes and uses a semi-bfs (bredth first search) based approach to obtain set of possible routes. Following this, based on some optimality parameters routes are filtered to obtain the ones with the minimal number of switches and intermediate stations involved. The route calculation methodology is defined in further detail in section 4 below. If there exist multiple optimal routes of travel for two entered stations, all of them are output after filtering rules are executed. This is shown in the case of the below illustrated example -



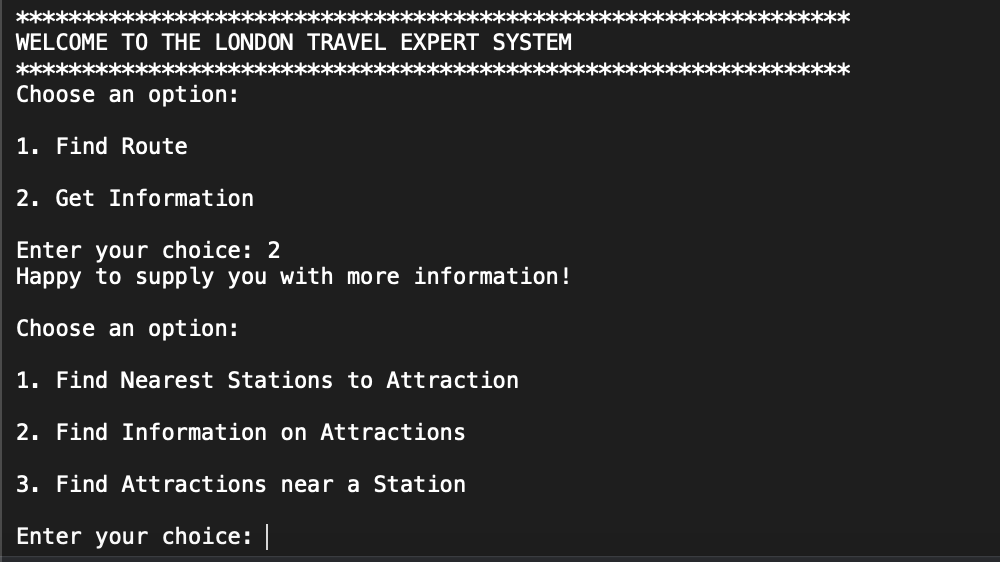
**Figure 3: Existence Of Multiple Optimal Routes**

In case, the user enters stations that do not exist on the metro line or in case the user makes a spelling mistake, an error rule runs and a message is displayed. This is shown below -



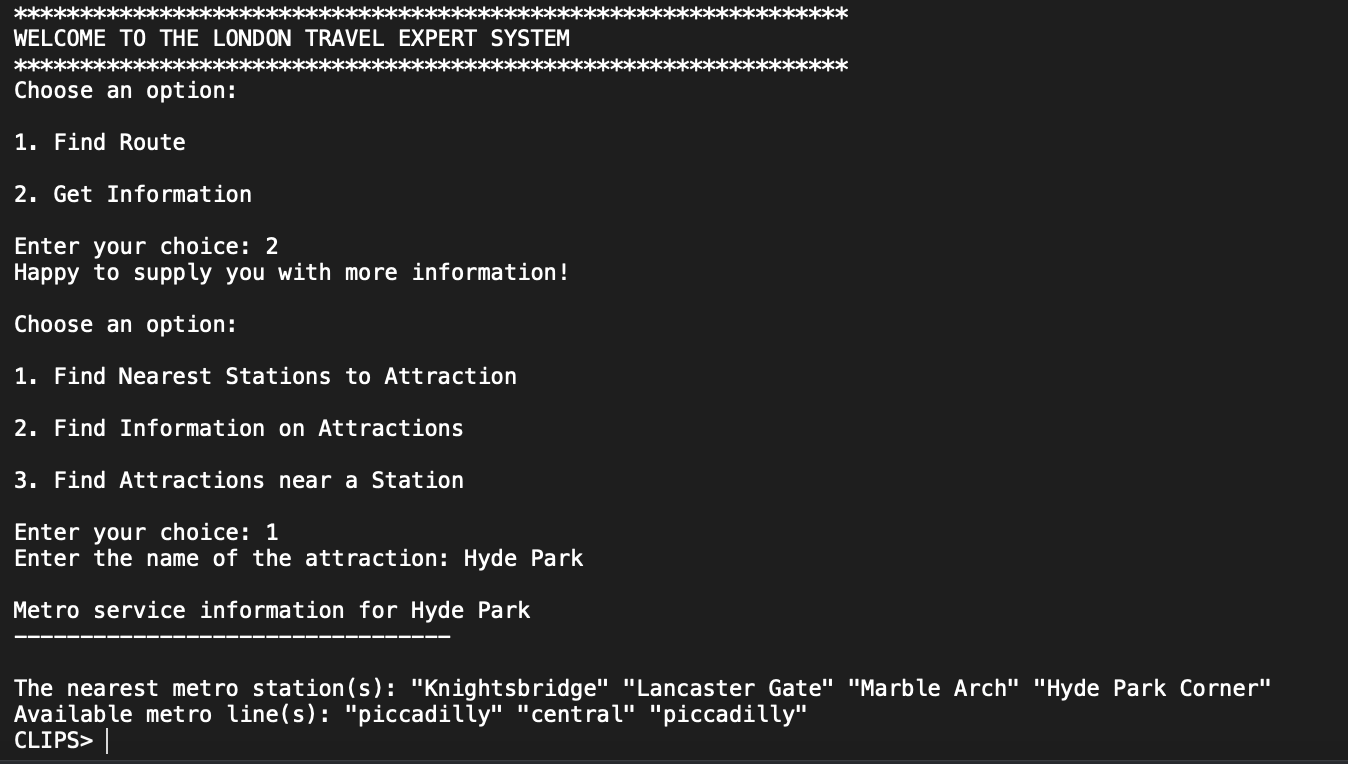
**Figure 4: Error Message For Spelling Errors And Non-Existant Stations**

Next, in case the user does not want to calculate the route but wants information on attractions in and around london itself, the second option, Get Information, can be used. Selecting this option produces another menu which shows the user the different types f actions they can perform to get information from the system. This is hown in the figure below -



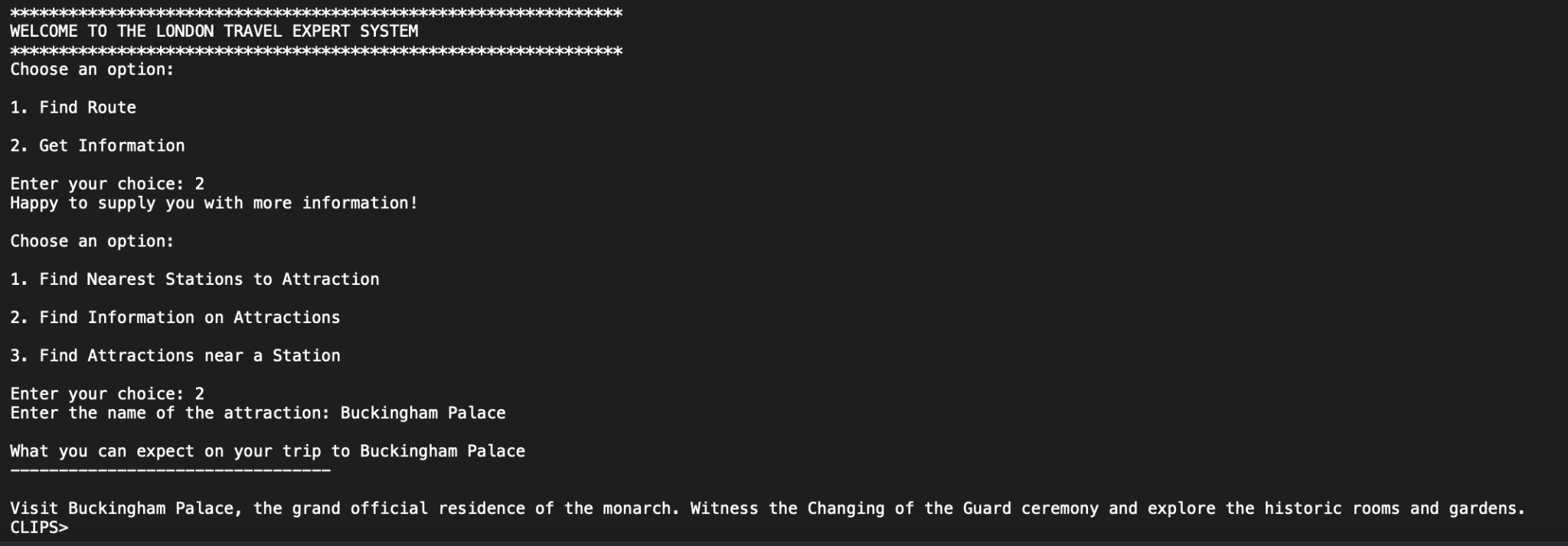
**Figure 5: Option Two - Finding Information On Attractions**

The first sub-option, when selected, prompts the user to enter the name of an attraction and based on fact matches according to defined rules, outputs the nearest metro stations to the attraction and the lines on which these metro stations are. This is shown in the example in the figure below-



**Figure 6: Getting Location Information about London Attractions**

The second option of the get information sub-menu prompts the user to enter the name of the attraction and outputs a brief description of the entered attraction. This is hown in the figure below -



**Figure 7: Getting Attraction’s Description from Name**

Finally, the last option in the sub-menu allows the user to enter the name of a train station on the London Tube System and in return outputs a list of all attractions near this station in a list format arranged in increasing order of distance within an acceptable radius. An example of this is shown in the figure below -



**Figure 8: Getting Nearest Attractions to a Given Station**

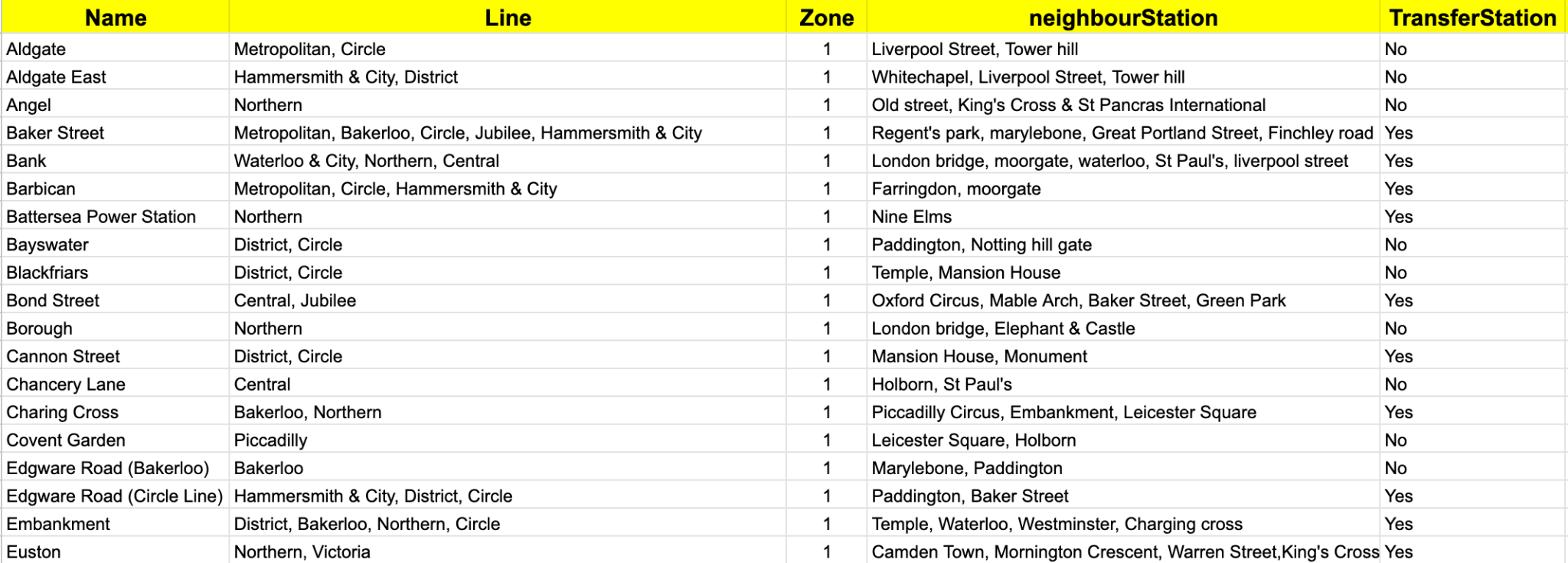
Thus, the created London Tube navigationsystem offers a menu driven user interface that allows the user to interact with the knowledge base of the expert system and obtain useful functionality from it. While this section served to provide examples of usage, in the following sections we describe the process of collecting data and crafting facts, rules and functions to obtain the final system.

**4 - Templates, Facts and Data Collection**

**4.1 Data Collection**

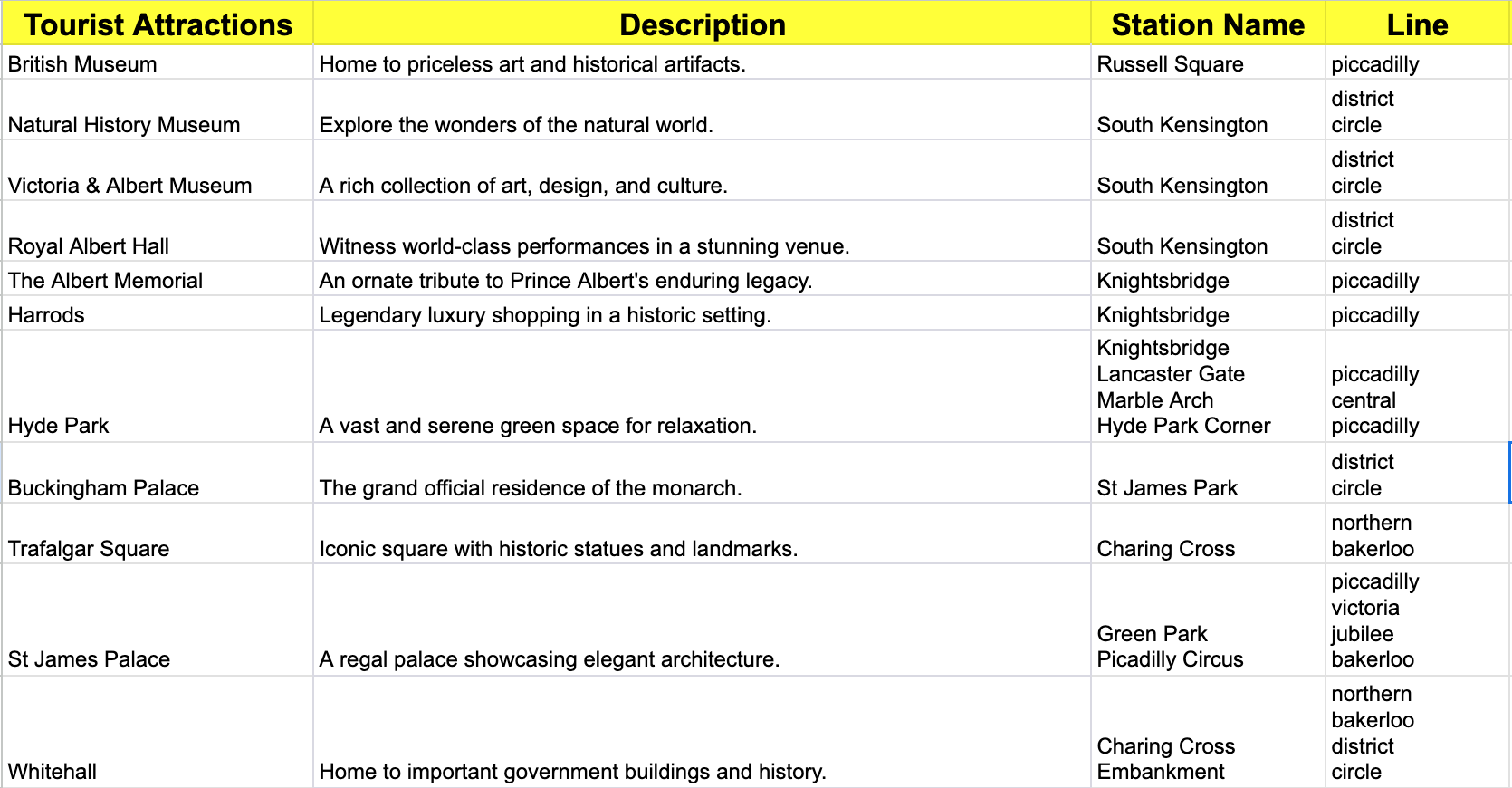
This section serves to briefly describe the identified entities that would form the knowledge base for our expert system and the data collection process to create facts on top of these entities. After careful and detailed analysis of the underground London tube system in the phase one of exert system development, we identified 4 key entity categories to help us model our expert system and perform key operations such as route and fare calculations. These were - station, lines, attractions and fares.

The station entity corresponds to information about the 121 stations of the London tube system in zone 1 and 2 that we are considering for our expert system development. The template for this entity has slots such as station name, line, zone, etc. In order to collect the data to populate the facts for this template, a list of the 121 stations under consideration was obtained from an online repository [1] by using python programming language to perform web scraping. Following this, the Transport for London Unified API or the TfL API [2] was used to collect additional information such as the lines that pass through a station, the neighbors of a tube station, etc. Figure 1 below illustrates a spreadsheet containing the data to populate facts for the station template -



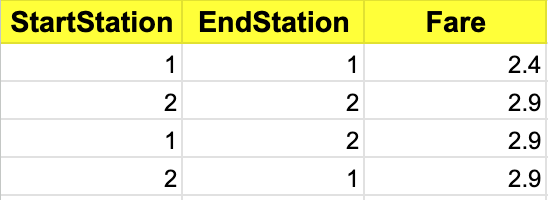
**Figure 9: Data to create facts for stations template**

For the attractions entity, the template contains slots such as attraction name, description, etc. To create facts for the entity’s template, a list of the 42 most relevant tourist attractions was obtained and then the Geocoding Google Maps API [3] was used to find the nearest relevant tube station for the attraction. The line information for the identified station was obtained by cross referncening already collected stations data as described at the beginning of this section. Python programming language in the jupyter notebook environment was used to achieve this. Furthermore, to obtain thedescription for the attractions, the Google Places API [4] was used. Figure 2 below illustrates a spreadsheet containing the data collected using the aforementioned procedures to populate the facts -



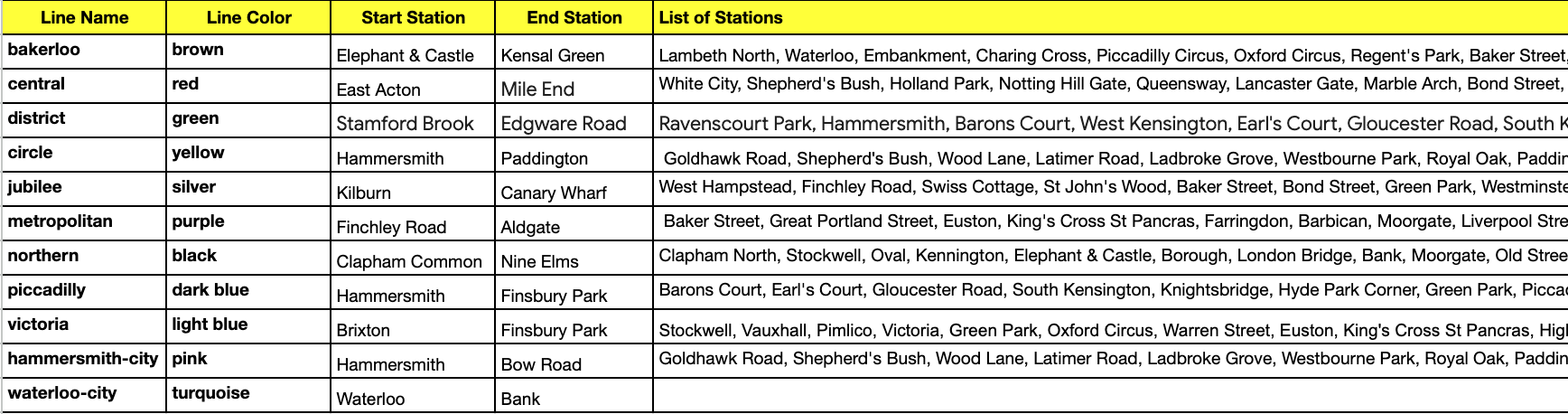
**Figure 10: Data to create facts for attractions template**

Thirdly, for the fare entity, the template will be populated with four facts only because we are considering only 2 zones for our expert system. The fares information included in the fact file is obtained from the fare table included in the project description document. Figure 3 below illustrates the spreadsheet containing fact data for fare template -



**Figure 11: Data to create facts for attractions template**

For the final lines entity, the created template has information such as the line name, the start and ending stations, the list of all stations, etc. This information was collected using the Transport for London Unified API or the TfL API [2] and a snapshot of the collected facts is shown in figure 4 below -



**Figure 12: Data to create facts for lines template**

The entities described above were identified during phase one of project development. However, after detailed requirements analysis, some changes and modifications had to be made to the templates and entities. One of the major ones was the removal of the fares template and addition of the switch template described in the following paragraph. More details on changes made to facts and templates in the current phase 2 of project development are presented in section 4.2 below.

One of the key entity for our system is the switch template and the corresponding facts. The switch template, which is described in detail in the following sections, was obtained from the data in figure 9. The stations which fell under the category of transfer stations were extracted from this data along with the lines that could be switched at them. Following this, for each of the lines, the other lines that could be reached from them via switching stations were computed using a modified bfs (bredth-first-search) algorithm. The pseudocode of this function is provided below -

function bfs\_find\_alternative\_paths(graph, start, end, max\_switches=4, max\_alternatives=3):

queue = create\_empty\_deque()

visited = create\_empty\_set()

alternatives = create\_empty\_list()

# Enqueue the starting node with initial paths and switches

queue.enqueue((start, [], [start], 0))

# Explore the graph using BFS

while queue is not empty:

node, edge\_path, node\_path, switches = queue.dequeue()

# Check if the destination node is reached

if node == end:

alternatives.append((edge\_path, node\_path))

# Check if the maximum number of alternatives is reached

if length(alternatives) >= max\_alternatives:

return alternatives

# Mark the node as visited

visited.add(node)

# Explore neighbors

for neighbor, edge\_name in graph[node].items():

if neighbor not in visited and switches < max\_switches:

new\_edge\_path = copy(edge\_path)

new\_node\_path = copy(node\_path)

new\_switches = switches + 1

# Update paths and enqueue the neighbor

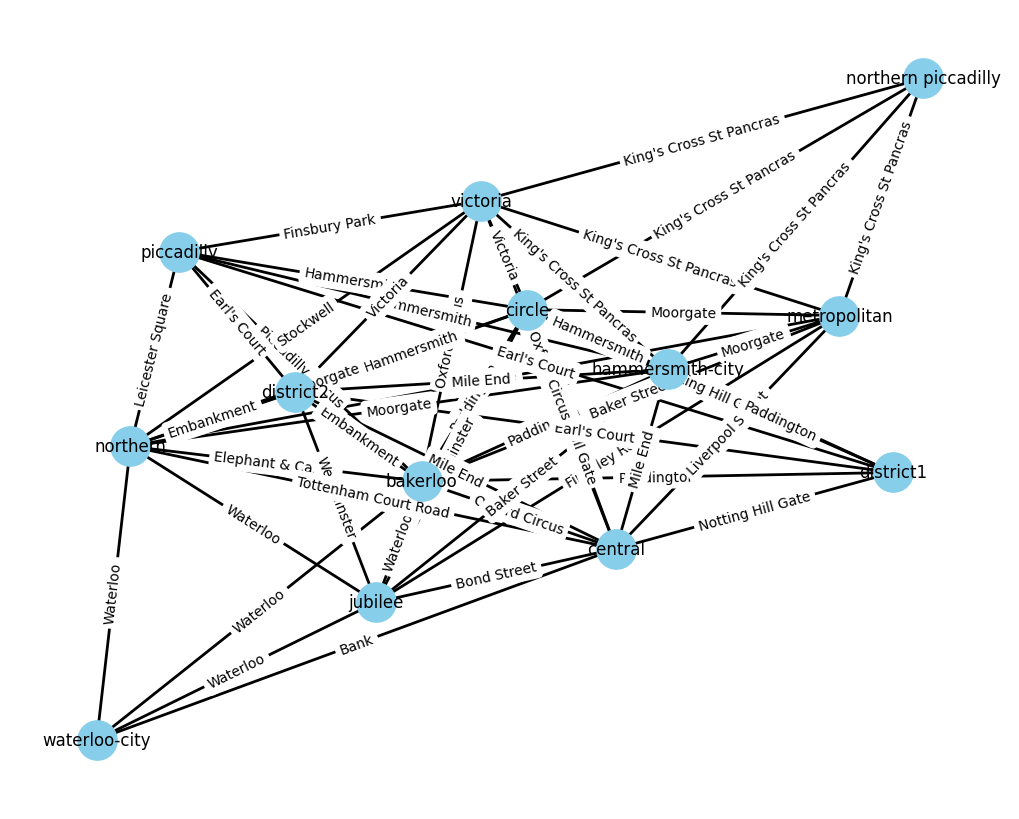
new\_edge\_path.append(edge\_name)

new\_node\_path.append(neighbor)

queue.enqueue((neighbor, new\_edge\_path, new\_node\_path, new\_switches))

return alternatives

Instead of using bfs to calculate the route between stations on the fly, the possible switches between lines are anticipated, recorded and fed into the knowledge base as facts using the above approach. This information was generated using the afiremetioned data and python programming language and the entire code for this can be found in the iPython Notebook [here [8].](https://colab.research.google.com/drive/1a3qAhu9FnyhL9GJln70Z37V9KBcY0Cxz#scrollTo=-pP8LE1BWJT5) The figure below illustrates a graph of switches and connections between the lines of the London Tube that was traversed to get the switch stations data.



**Figure 13: Data to create facts for lines template**

Using the methods described in this section, suitable data was gathered and was structured into templates and facts in order to create the knowledge base of our expert system. The following sections provide more detail on this process as well as the modifications and changes made from phase one.

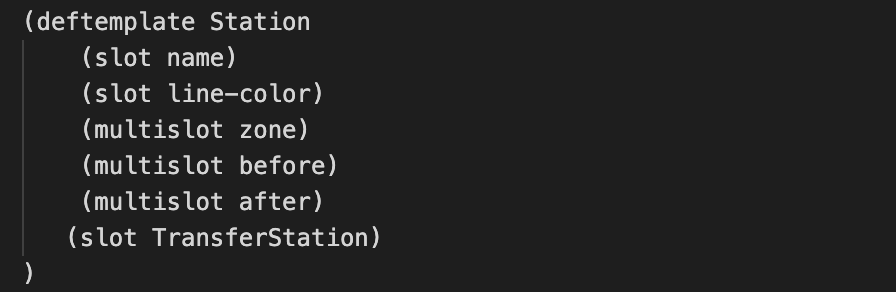
**4.2 Revisions in Templates and Facts**

Templates are a fundamental concept in Clips and are used to define the structure of facts. They serve as a blueprint for creating instances of facts. Previously our system had the following templates: Station, Fare, Line, and Attractions. As we proceeded further with developing a rule based system for getting routes and attraction information, the Station template was changed to store a list of inbound and outbound stations. Fare and Line facts were not needed so instead added a fact for Switch station and AttractionInfo which will be described further. Finally the system has following templates: Station, Switch, Attractions, and AttractionsInfo. These are described in the sections that follow.

**4.3 Templates in Clips:**

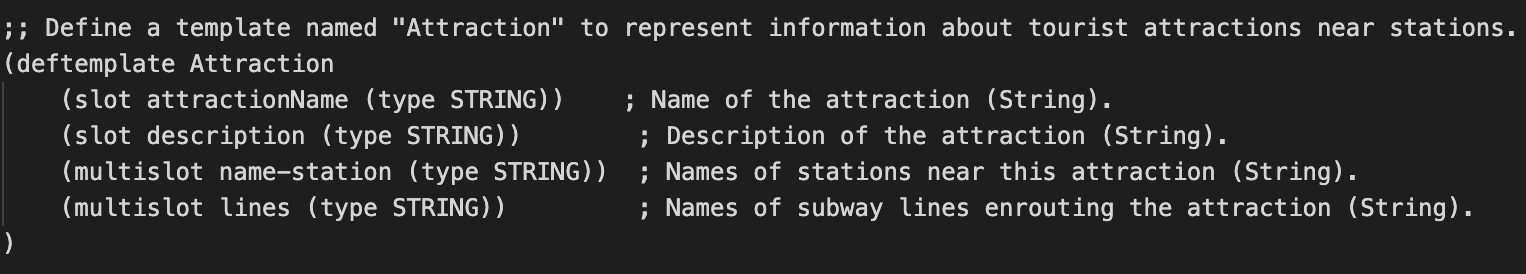
Below is the list of templates created :

* **Station:** This template is designed to capture various attributes of a station like:
  + Name: This slot is used to store the name of the station, which is of type STRING, allowing for text-based representation.
  + Line: This is a multislot, which means it can hold multiple values. It is used to store the names of the subway lines that enroute this station.
  + Zone: This is also a multislot and it stores integer values indicating the zone the station it lies in.
  + Stations Before: The "before" is a multislot for storing the list of all the stations before the current station
  + Stations After: The "after" slot is a multislot for storing the list of all the stations after the current station
  + TransferStation: This slot is also of type string and it stores yes or no depending on whether the station is transfer station or not.



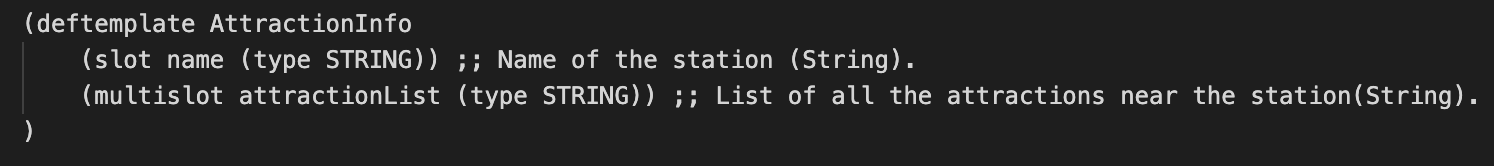
**Figure 14: Creating a template for Station entity**

* **Attractions:** This template is designed to capture various attributes of line attractions:
  + Attraction name: This slot used to store the names of attractions in zone 1 and zone 2, which is of type STRING.
  + Description: This slot used to store the description of each attraction , which is of type STRING.
  + Station name: This multi slot used to store the names of stations nearest to this attraction, which is of type STRING.
  + Line: This multi slot used to store the names of lines that are connected to the nearest station from this attraction, which is of type STRING.



**Figure 15: Creating a template for Attractions entity**

* **AttractionInfo:** This template is designed to capture list of attractions against each station:
  + Name: This slot is used to store the name of the station, which is of type STRING, allowing for text-based representation.
  + Attraction List: This multi slot used to store a list of attractions near the given station.

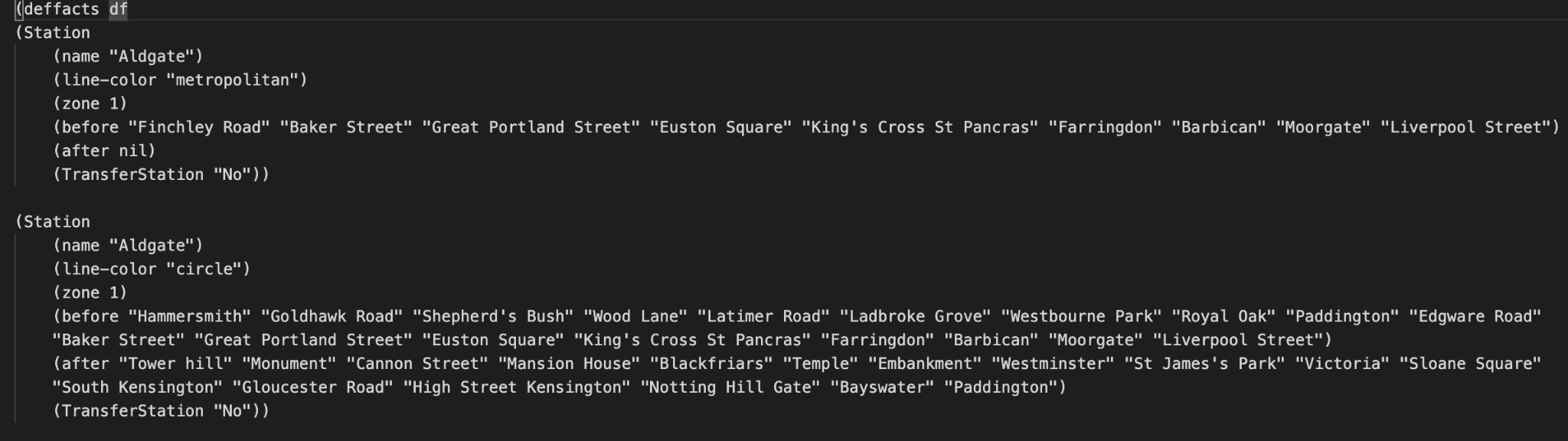


**Figure 16: Creating template for Attraction List against Stations**

**4.4 Facts in Clips**

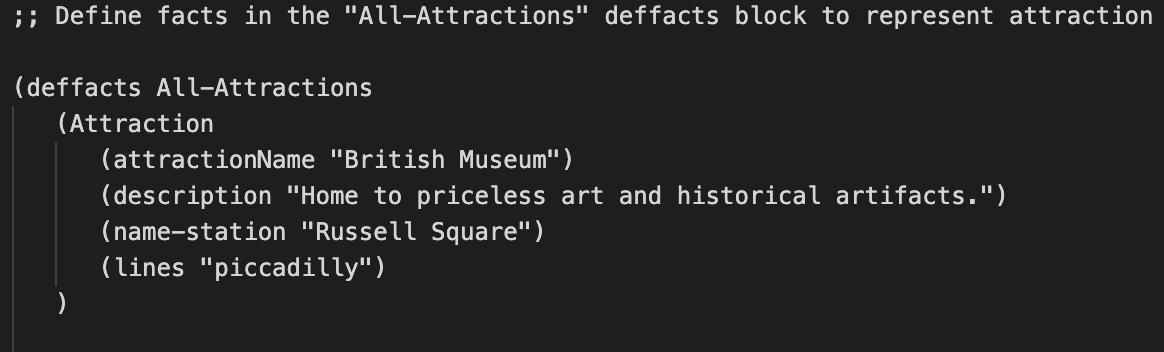
Facts are instances of templates. Facts are created by asserting them into Clips' working memory using the assert command, followed by the template name and values for each slot. They can also be defined using deffacts to define and initialize facts within the CLIPS working memory. They provide a convenient way to assert predefined sets of facts. Below are attached screenshots for instance of each facts created in our system:

* **Station Facts:** They are structured in “station-facts” block to represent name of each station, lines that enroute each station, zone it lies in, the neighboring station this station and if its a transfer station or not.



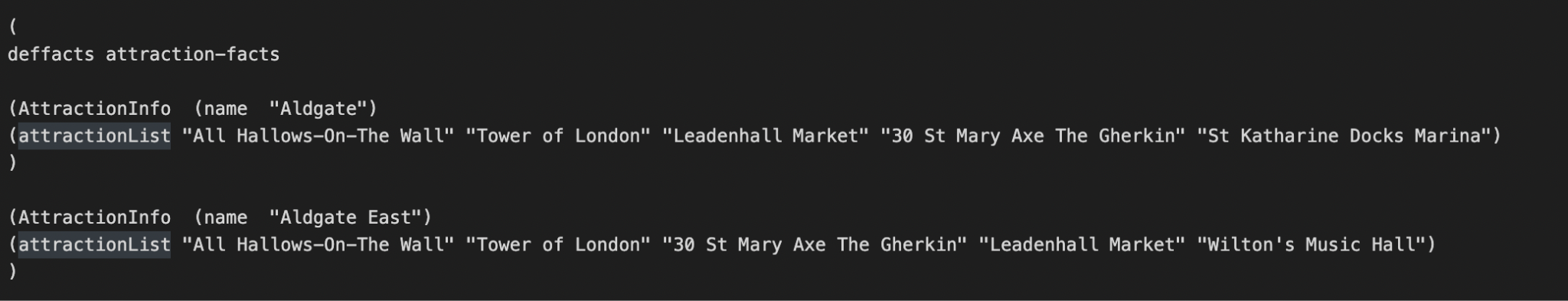
**Figure 17: Adding facts for Station template**

* **Attractions Facts:** They are structured in "All-Attractions" deffacts block to represent attraction name, its description, the nearest station to that attraction and lines that enroute the nearest station, for each attraction in zone 1 & zone 2 based on the template defined.



**Figure 18: Adding facts for Attraction template**

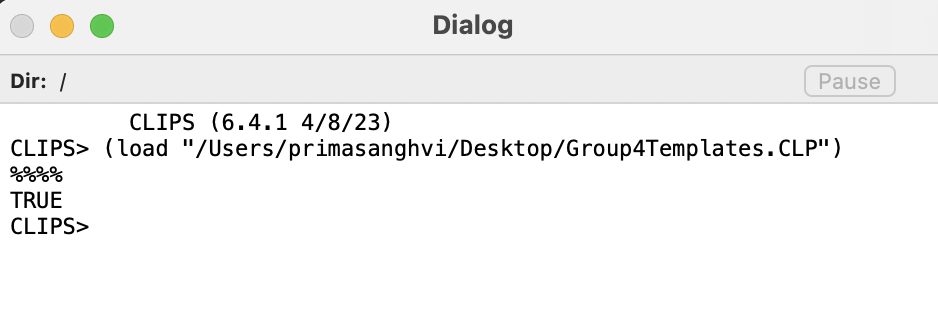
* **AttractionInfo Facts:** They are structured in "attraction-facts" deffacts block to represent station name and list of all attractions near it, for each station in zone 1 & zone 2 based on the template defined.



**Figure 19: Adding facts for Attraction template**

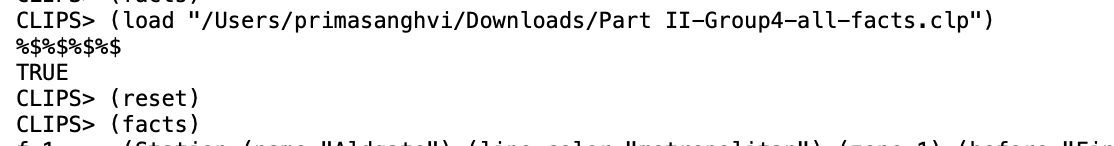
**4.5 Loading Facts and Templates in Clips**

On the clips IDE template and facts file which has .clp extension can be directly loaded using the (load) command followed by the file path in parentheses.



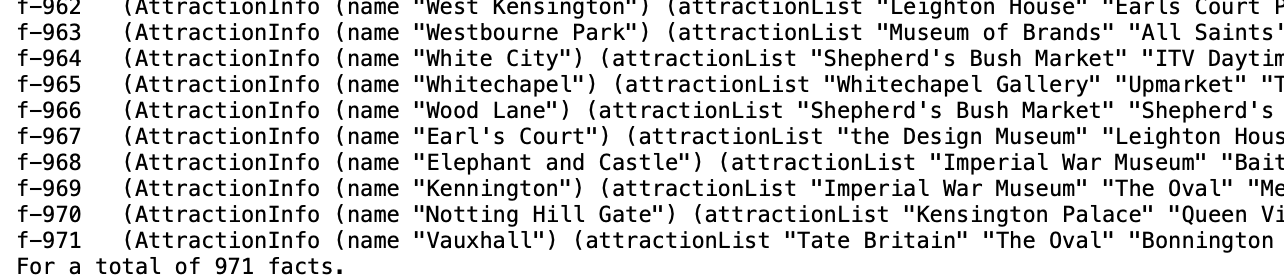
**Figure 20: Starting the CLIPS environment**

It returns true once the template is loaded and saves it in memory. Now the facts are loaded in a similar manner using the (load) command followed by the file path in parentheses. After loading the facts give **(reset)** command followed by **(facts)** command to view all the facts that are saved in memory.



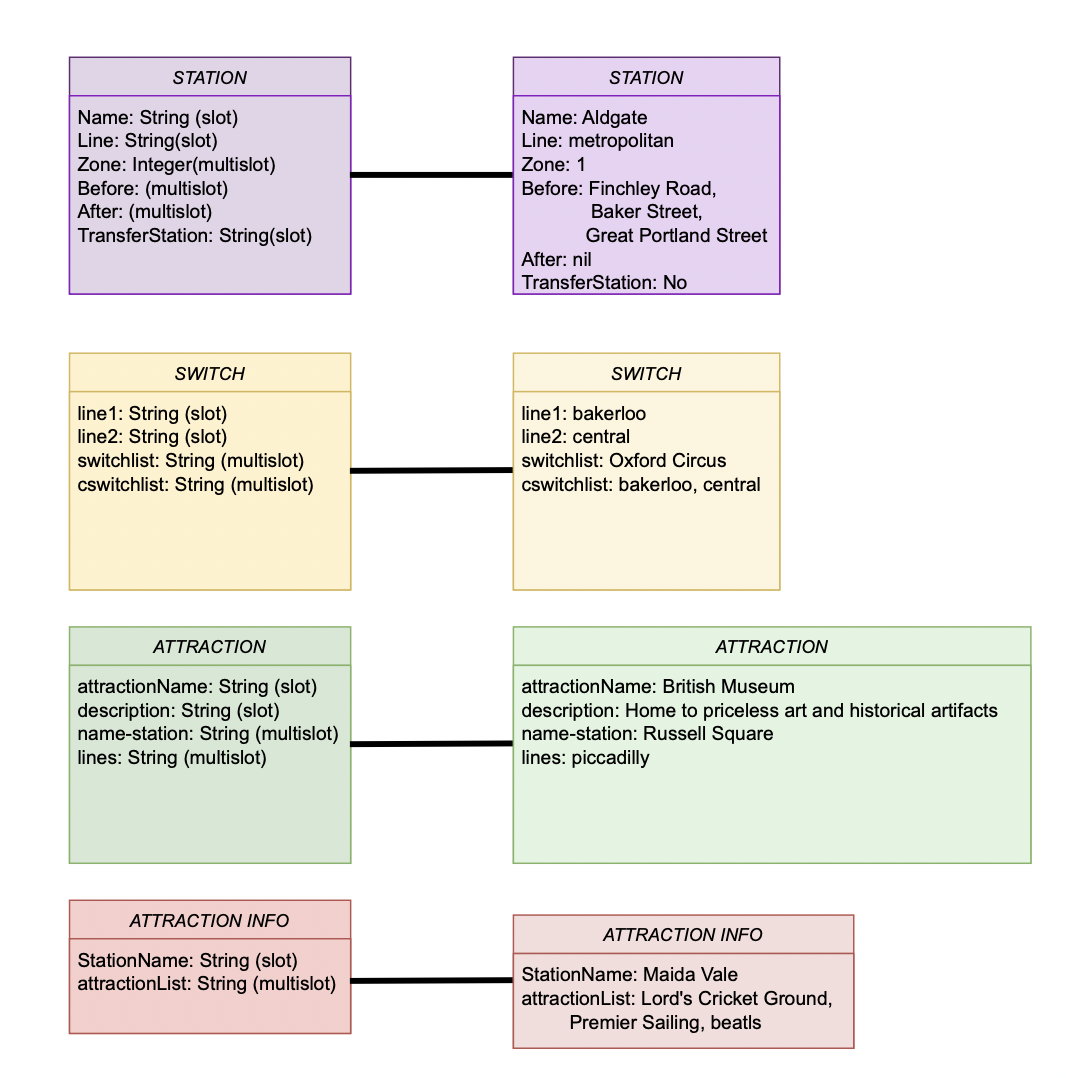
**Figure 21: Loading facts into CLIPS**

There are total of 971 facts as seen in below screenshot -



**Figure 22: Viewing the entered facts in CLIPS**

The diagram below gives a high level illustration of the facts, their slots and example facts created for each of the facts -

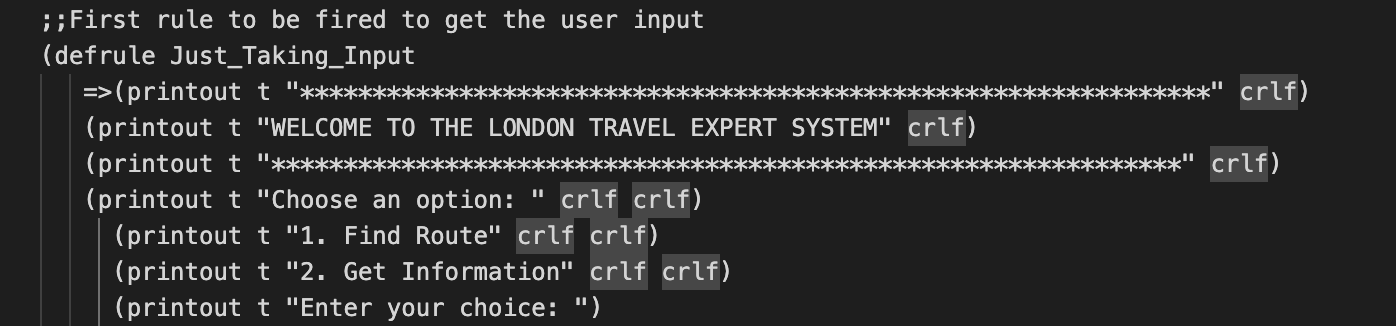
****

**Figure 23: Diagrammatic Illustration of Templates and Example Facts**

**5 - Design and Implementation of Rules**

CLIPS is a system development tool used to create rule-based systems. Rules are structured with condition and the action part. The action part contains the tasks to be executed when the condition is satisfied. The condition part of a rule uses pattern matching to identify specific patterns in the data or facts already stored within the system. In our system following rules are defined to get routes and information on attractions in London. The first rule fired is for taking user input based on the choice given.

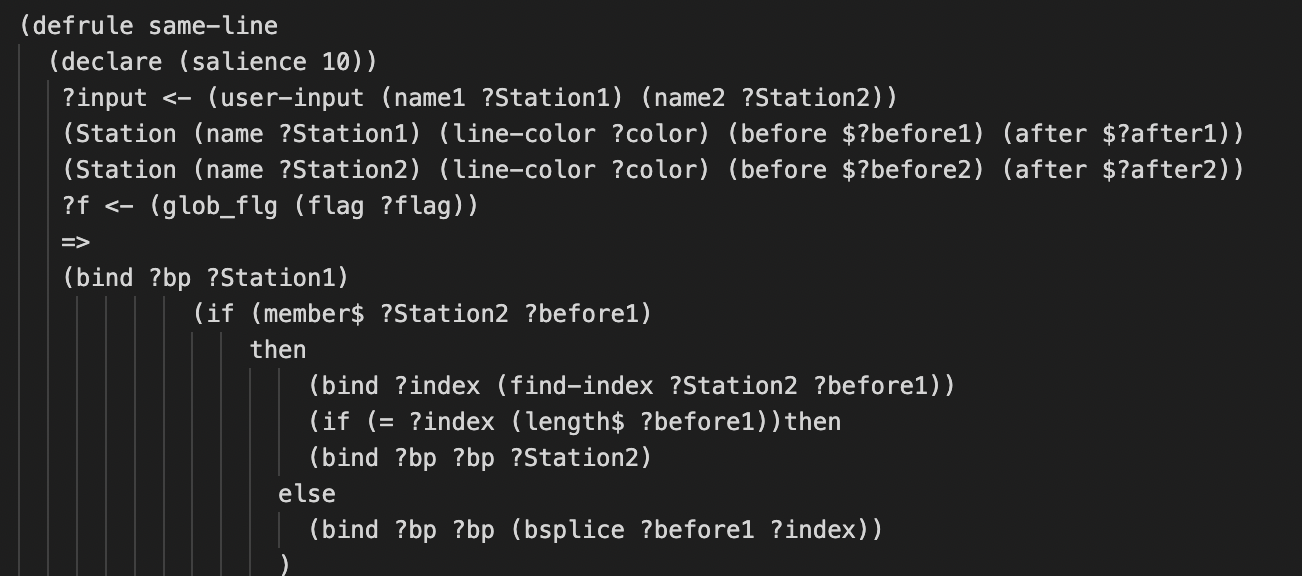
* **Input rule:** It prompts the user with a multiple choice on whether he/she wants information on station or route between start and destination station. According to the user entered choice it fires another. Below is the screenshot for the same.



**Figure 24: User Input Rule**

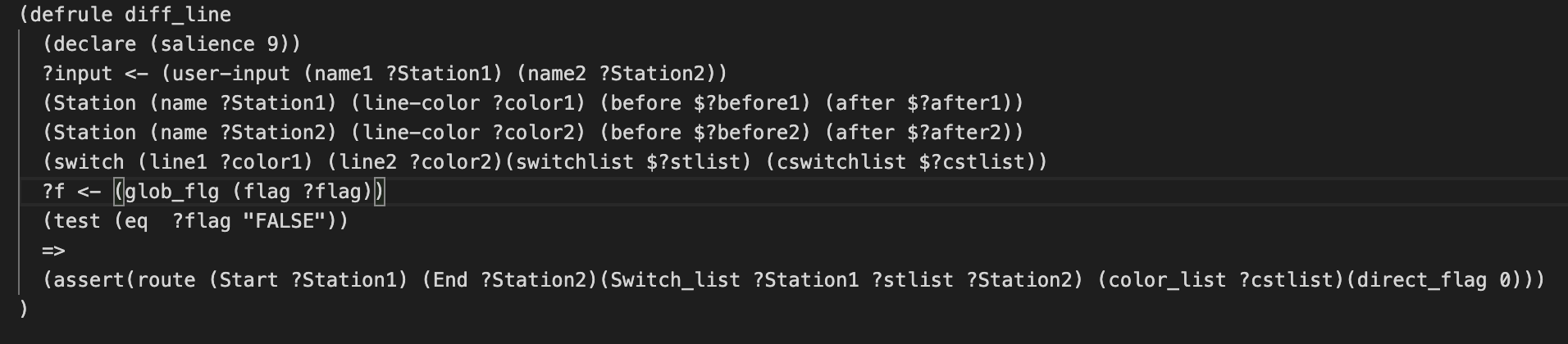
Next, if the user selects choice one the route rule is fired -

* **Routes:** This rule takes start station and end station as input and calculates all the possible routes between them. First it checks in the station facts if both the entered stations are on the same line and returns the list of stations in order from start to end. The screenshot of that rule is as below:



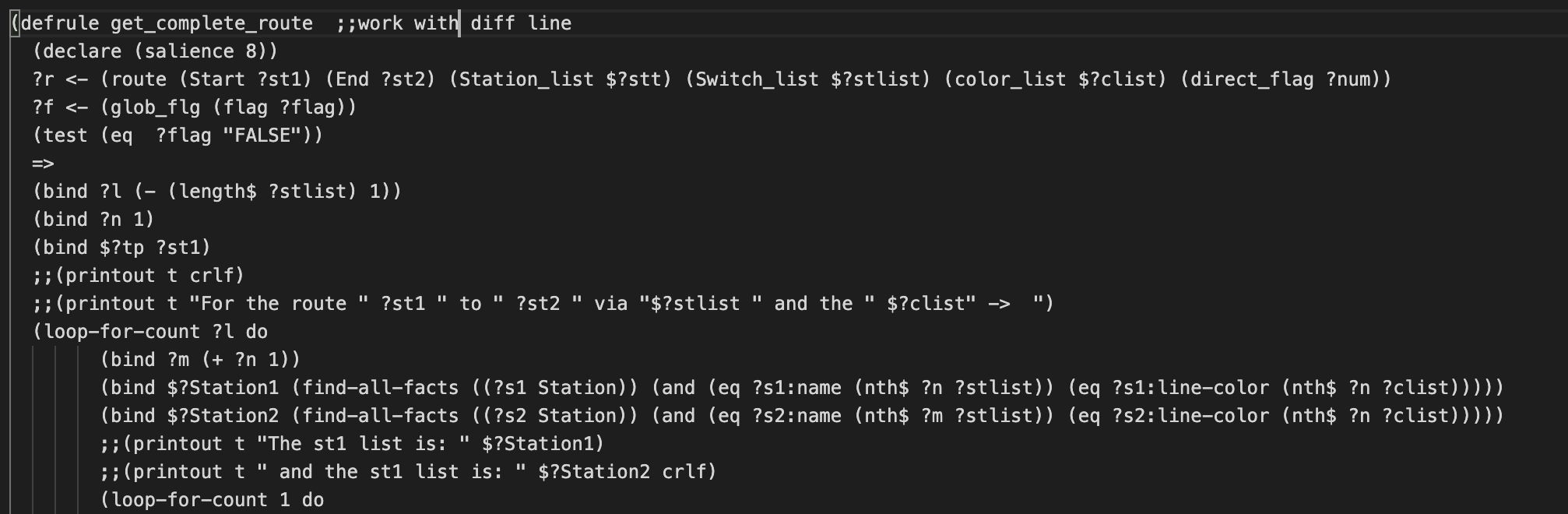
**Figure 25: Rule for stations on same line**

Further, if they are not on the same line it fires the diff\_line rule to assert new facts containing all possible routes based on the stations and switch facts. The code screen shot for same is as below:



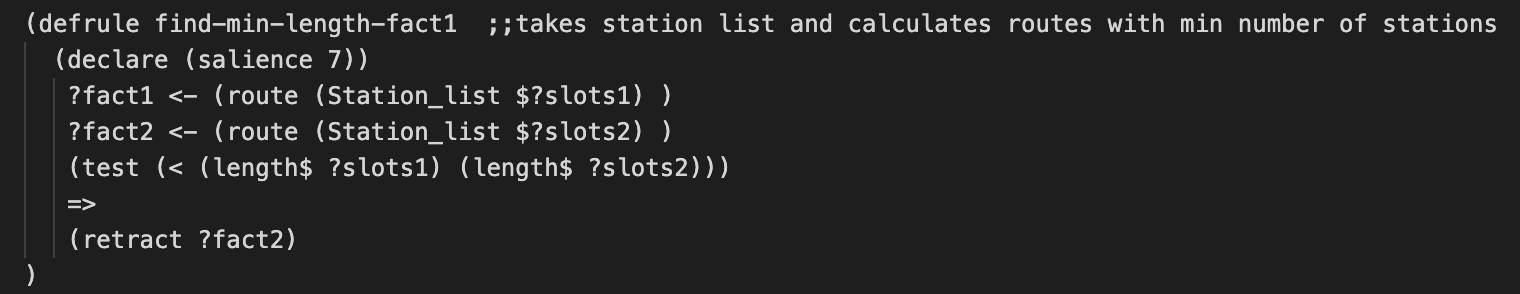
**Figure 26: Rule for stations on different lines**

Now once these facts are asserted into the system get\_complete\_route will calculate the optimal route and return it to the user. In order to calculate the optimal route it will retract all the facts that have a minimum number of stations and eliminate the rest. Now if there is more than one route with minimum stations; in that case it will check for the number of line switches and select the one with minimum switches. And in the end if there are again more than one possible route it will prompt all the possible options. Instead of going through standard BFS we choose this method for optimization to handle the features of phase 3. Below is the screenshot of the same:



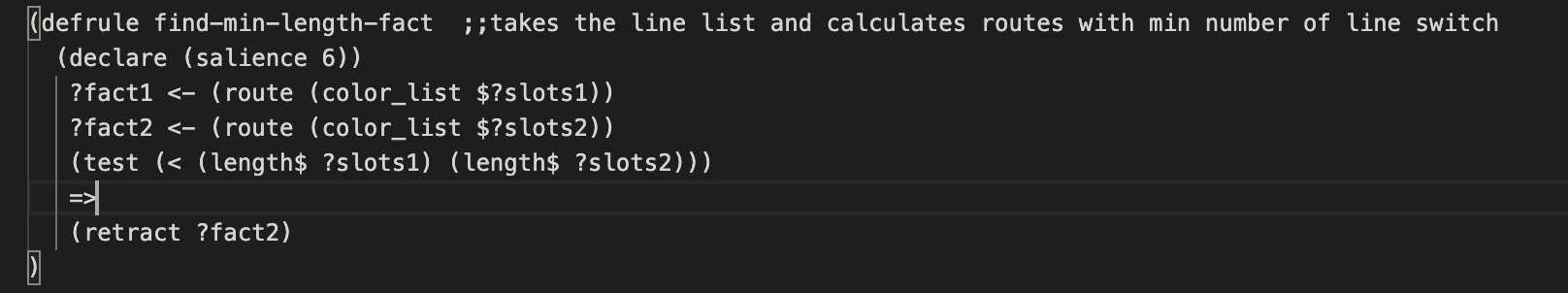
**Figure 27: Main Rule to Calculate Routes**

In addition to these, find-min-length-fact1 will take a station list and calculate the routes with the minimum number of stations.



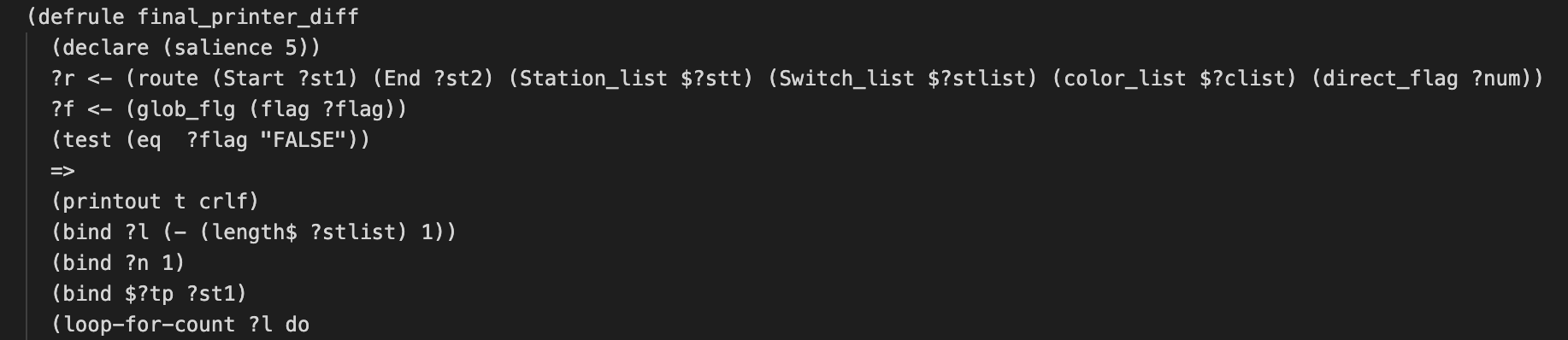
**Figure 28: Rule to calculate min stations**

The find-min-length-fact rule is used to take the line list and calculate routes with a minimum number of line switches.



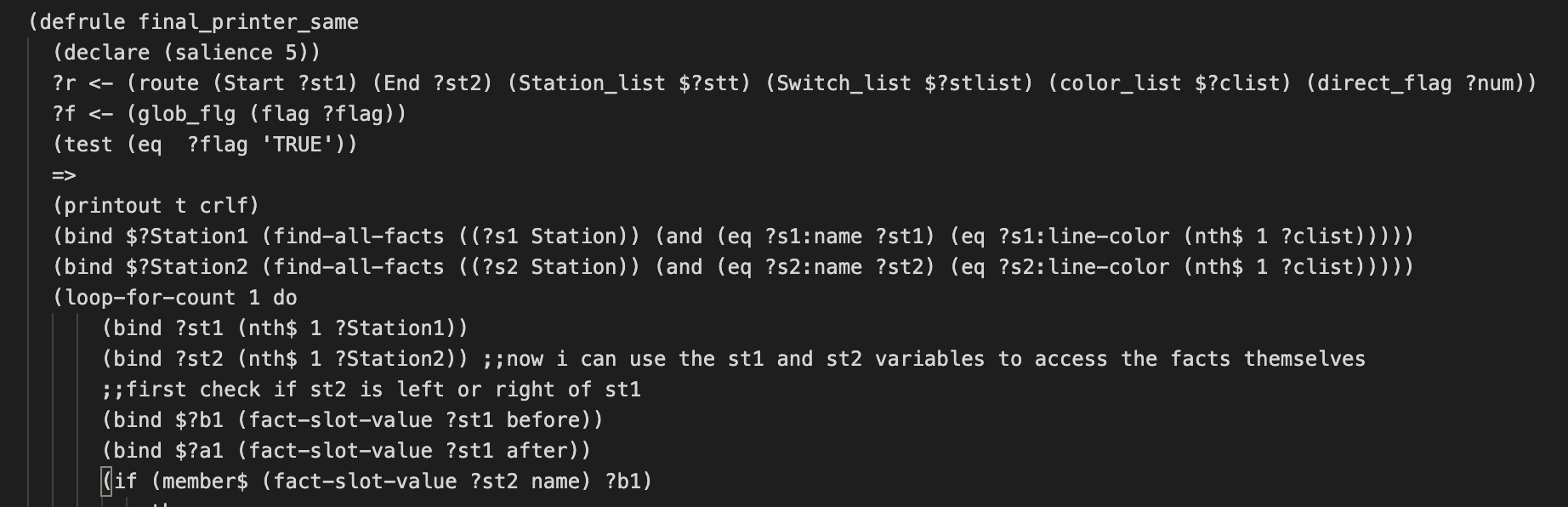
**Figure 29: Rule to calculate min line switch**

Now, to print the output of the route final\_printer\_diff will print the output for the case when start and end stations are on different lines.



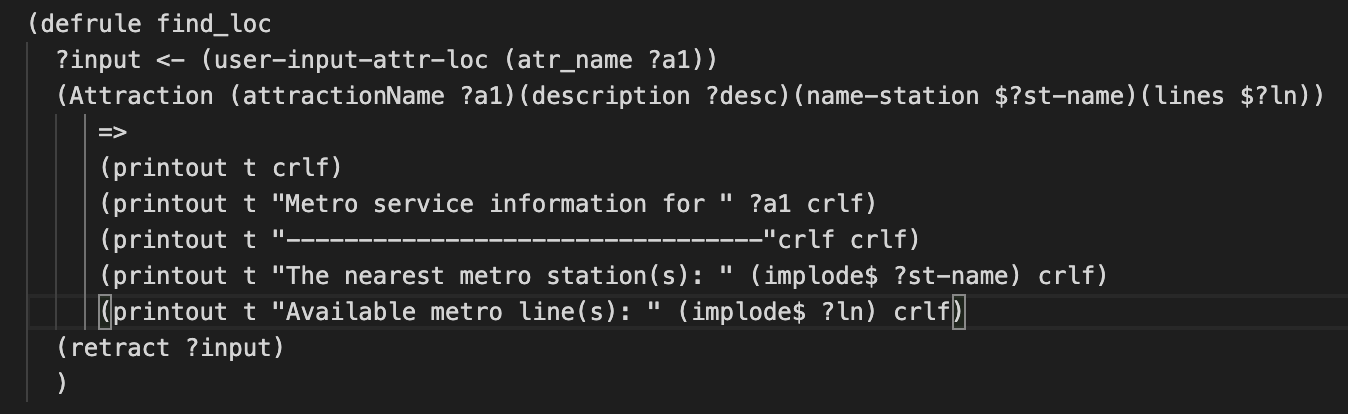
**Figure 30: Rule to print output when different lines**

Now, to print the output of the route final\_printer\_same will print the output for the case when start and end stations are on the same line.



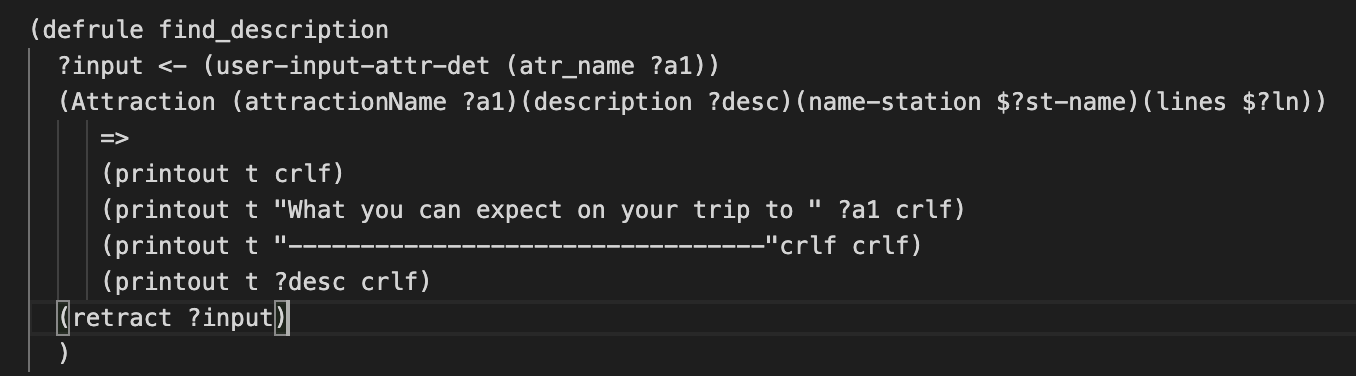
**Figure 31: Rule to print output when same line**

* **Attractions:** This has three more rules based on the option selected. When the user selects get information option it will again prompt 3 options as: Find Nearest Stations to Attraction, Find Information on Attractions, and Find Attractions near a Station. If user selects option 1 it will fire find\_loc rule and based on the Attraction fact fetch details on nearest metro station to the attraction entered by the user.



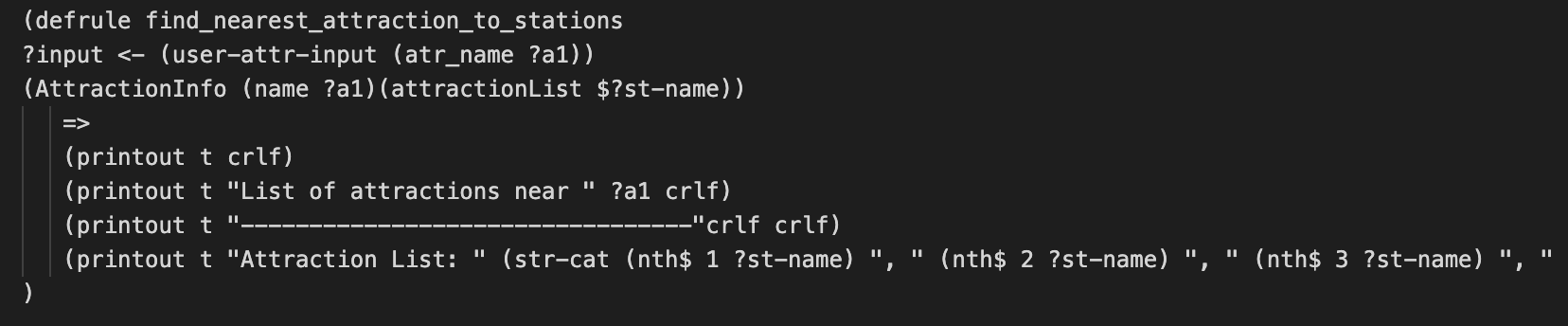
**Figure 32: Rule to find metro station closest to the attraction**

For option 2 it will fire the find\_description rule and based on Attraction fact it will fetch the description of the attraction entered by the user.



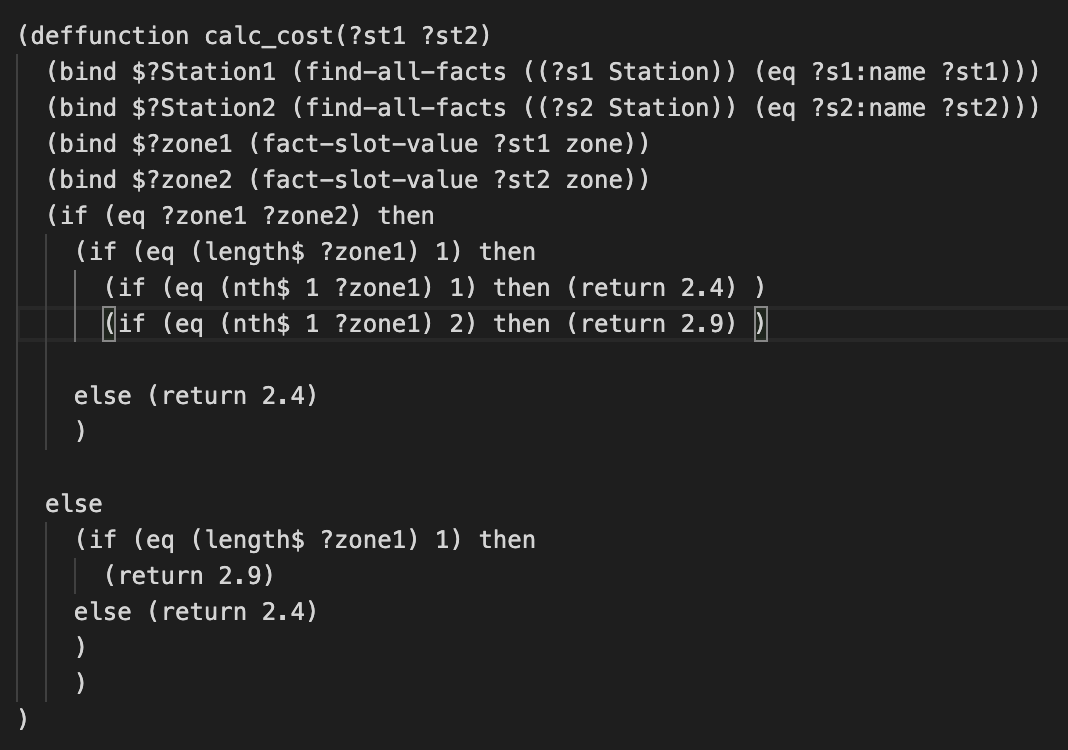
**Figure 33: Rule to find attraction details**

For option 3 it will fire the find\_nearest\_attraction\_to\_stations rule and based on the AttractionInfo fact it will fetch the list of top 5 attractions near the entered metro station.



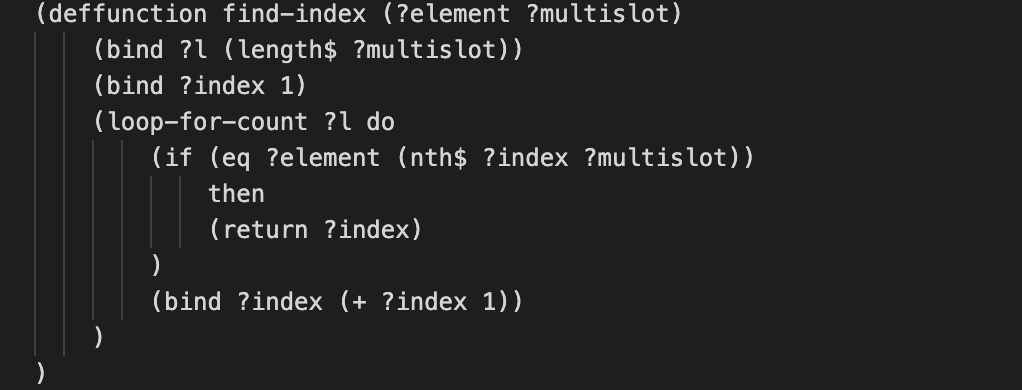
**Figure 34: Rule to find closest attractions near metro station**

* **Functions:** Apart from rules some functions have been implemented. They are as follows:
  + **Calc\_cost:** This function calculates the cost of travel from one station to another in route based on zone positioning.



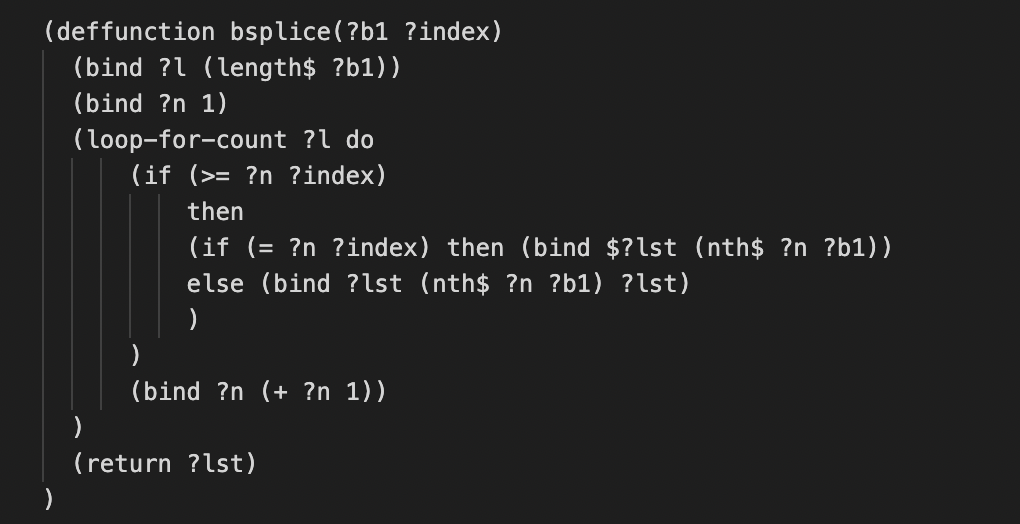
**Figure 35: Calculate cost function**

* + **Find\_index:** Calculates the position of station in the before and after list of another station



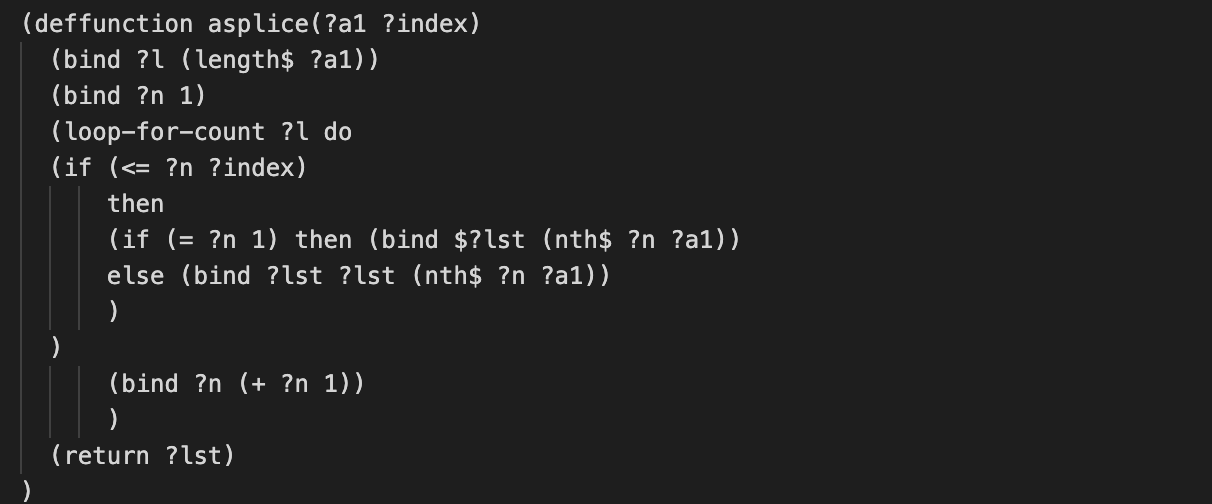
**Figure 36: Find Index Function**

* + **Bsplice:** Calculates the subset of stations from the before travel list that needs to be traveled to get to the destination station.



**Figure 37: Function to calculate the subset of stations from before travel list**

* + **Asplice:** Calculates the subset of stations from the after travel list that needs to be traveled to get to the destination station.



**Figure 38: Function to calculate the subset of stations from after travel list**

**6 - Conclusions and Future Scope**

After identifying entities, defining templates and populating facts based on information collected from diverse sources in this phase of the expert system development for navigating the London tube system, we designed and implemented rules for developing rule-based systems and knowledge-based applications. We developed an interactive system which will give information on routes and attractions in London. Apart from rules, functions are also created for calculating the cost or trip fare and some intermediate functions to calculate the final optimized route from source to destination.

In the next phase, our focus will be on developing a GUI system to enhance the functionality of our system. Further, we also plan on building on the hybrid-bfs line switch mechanism to handle out of service stations during phase three of project development.

**7 - References**

[1] Wikipedia contributors. (2023, September). List of London Underground stations. Wikipedia. URL: https://en.wikipedia.org/wiki/List\_of\_London\_Underground\_stations

[2] Transport for London. (2023, September). TfL API. URL: https://api.tfl.gov.uk/

[3] Google. (2023, September). Google Maps Geocoding API. URL: https://developers.google.com/maps/documentation/geocoding/start

[4] Google. (2023, September). Google Maps Places API. URL: https://developers.google.com/maps/documentation/places/web-service/overview

[5] *CLIPS User Guide Chapter 2*. (n.d.). https://redirect.cs.umbc.edu/portal/clips/usersguide/ug2.html

[6] GeeksforGeeks. (2023, June 9). *Breadth first search or BFS for a graph*. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[7] Robinson, P. (n.d.). *CLIPS Tutorial 1*. <https://kcir.pwr.edu.pl/~witold/ai/CLIPS_tutorial/CLIPS_tutorial_1.html>

[8] Chadha, H. (2023, November 7). *BFS for London Tube Navigation System*. Google Colaboratory. https://colab.research.google.com/drive/1a3qAhu9FnyhL9GJln70Z37V9KBcY0Cxz#scrollTo=-pP8LE1BWJT5