Restaurant Expert System

LBA

Minerva University

CS152

Professor Shekhar

December 12, 2022

Restaurant Expert System

1. Aims

For this LBA project, we selected the first option, therefore, we will be focusing on an expert system to recommend restaurants to hungry Minervan students. To better understand the problem, we proactively applied the #rightproblem HC.

1. 1. Initial State

The initial state of our system is a case in which Minervan students do not currently have access to software that can narrow down restaurant recommendations from a set of filter parameters that are idiosyncratic to the Minervan student schedule. While there exist services such as PedidosYa, Rappi, and Google Maps, in Buenos Aires, they lack curation, and leaving the system in its current state might lead to hungry, frustrated Minervas in Buenos Aires.

1.2. Goal State

The aim of the group is to fill this market gap by providing a curated restaurant recommendation expert system built by Minervans for Minervans. Our system is expected to offer detail and be tailored towards Minervans' preferences for factors like delivery preferences, language, and varying dietary preferences.

Figure 1. Database for expert system

Restaurant	Distance (walking)	Cuisine	Diet	Service Options	Languages	Cost	Rating	Atmosphere
Casa Munay	1 min	Western	Veg	Dine-in	Spanish	$$	4.5	Cosy
Koko Bao Bar	3 min	Asian	Non-veg	Dine-in	Spanish	$$	4.3	Casual
Mumbai Indian Food	8 min	Indian	Non-veg	Dine-in	English	$$	4.3	Casual
Ajo Negro	10 min	Italian	Non-veg	Delivery	Spanish	$$	4.5	Casual
Restaurant Montanes	11 min	Spanish	Non-veg	Dine-in	Spanish	$$	4.2	Casual
Chipper Seafood	12 min	Caribbean	Pescatarian	Delivery	English	$$	4.3	Casual
Anchoita	14 min	Argentinan	Non-veg	Delivery	Spanish	$$$	4.6	Casual
Georgie's	15 min	Mexican	Non-veg	Delivery	English	$$	4.6	Cosy
Restaurant Atlanta	22 min	Argentinan	Non-veg	Delivery	Spanish	$$	4.4	Casual
El Pollo Rico	22 min	Peruvian	Non-veg	Dine-in	Spanish	$$	4	Cosy

Figure 1 above displays the curated restaurant data. Rather than having an overwhelming amount of restaurants as on other platforms, we have tailored the top 10 restaurants close to the residence hall according to M24 students. Our 8 askables include: free time availability (free_time), ingredients the food should include (ingredients), dietary restrictions (diet), student's mood (mood), student's activity (activity), student language preferences (language), cost preferences (cost), and rating preferences (rating). A more detailed breakdown is provided within the data collection document which can be found here.

1.3. Scope

The scope of the project is limited to a Minervan audience, specifically Minervans in Buenos Aires as they are the most likely to benefit from this application. This is a well-adjusted scope that allows for a good level of curation to Minerva-specific needs.

1.4. Obstacles

There are a number of identified obstacles that stand between the starting situation and the goal state. These include

- Varying time zones that limit working hours within our team.
- No centralized database with the restaurant information is needed.
- Language barriers hinder the primary collection of restaurant data.
- Limited expertise with prolog.

1.5. Constraints

We are also under various constraints while solving this problem. Some of them include the limited time because of various assignments, finals week, and classes.

2. Data Collection

For the data collection, we decided to ask M24s on the group chat what their favorite restaurants were. This was an appropriate action to understand their wants for a better design. We also collected data from reputable local apps such as Rappi as well as TripAdvisor and GoogleMaps. Additionally, we conversed with the concierge for recommendations they have given other Minervans. After that, we selected the top 10 of them to include in our KB.

For more involved data columns such as the distance column, we utilized the Google Maps distance feature for accurate walk-time predictions. Similarly, for ratings and cost, we utilized the ratings from Google maps relying on the heuristic that a larger sample size offers accuracy and reduced noise.

For the English-speaking column, there was no information we could use online to base it on. The only sources we had were other people's experiences. If we had visited the restaurant and knew they spoke English, we would say that they are English-speaking. If we did not know, we asked other Minervans and looked at their reviews online to provide a more educated answer. This is not ideal but it is valuable information that Minervans like to have so we decided to include it.

2. **Visualization**

   Please reference Figure 2 in Appendix C for the visualization of the expert system.

3. **The Expert System**

   You can find all of our code in this Forum workbook or screenshots in Appendix D.

4. **Analysis**

These are the three test cases we covered.

- No restaurant matches a criterion.

```
Out [58]    free_time is very_low? no
            free_time is low? no
            free_time is high? no
            There are no restaurants in our database that match your description.
```

- Exactly one restaurant matches the criteria:

```
free_time is very_low? yes
ingredients is salads? no
ingredients is ketchup? no
ingredients is rice? yes
diet is non_vegan? yes
activity is hang_out? yes
language is spanish? yes
cost is low? yes
rating is very_high? yes
mood is chill? yes
Your restaurant option(s):
1) koko_bao_bar
```

- More than 1 match: we artificially added a duplicate of casa_munay for this test

```
free_time is very_low? yes
ingredients is salads? yes
diet is vegan? yes
activity is hang_out? yes
language is spanish? yes
cost is low? yes
rating is very_high? yes
mood is homey? yes
Your restaurant option(s):
1) casa_munay
2) casa_munay2
```

- Test the right-most branch (high free time) where meat can lead to 2 options:

```
free_time is very_low? no
free_time is low? no
free_time is high? yes
ingredients is meat? yes
diet is non_vegan? yes
activity is relax? no
activity is hang_out? yes
language is spanish? yes
cost is low? yes
rating is high? yes
mood is homey? yes
Your restaurant option(s):
1) pollo_rico
```

```
free_time is very_low? no
free_time is low? no
free_time is high? yes
ingredients is meat? yes
diet is non_vegan? yes
activity is relax? yes
language is spanish? yes
cost is low? yes
rating is very_high? yes
mood is chill? yes
Your restaurant option(s):
1) atlanta
```

Part 6: Real-life Application

The system performs well in real life. The variety of restaurants is not big right now and it does not allow for listing several categories for cuisine for example. We got there and it turned

out the place was closed so adding open hours and days would be another improvement. Please reference Appendix C for the group image.

Team Contributions

Philip: parts 1 and 2 (introduction and data collection), extension 1 and 2, appendix

Vicki: parts 3-6, references, appendix

Andri: none, we tried reaching out on telegram and email. Andri has not been online for the past four days so probably something came up for him.

References

Google. (n.d.). Buenos Aires map. Retrieved December 12, 2022, from https://www.google.com/maps.

Rappi. (n.d.). Restaurants. Retrieved December 12, 2022, from https://www.rappi.com.ar/restaurantes.

TripAdvisor. Restaurants. Retrieved December 12, 2022, from https://www.tripadvisor.com/.

Appendix A: HCs

#rightproblem - We utilize this HC in the problem definition to characterize the problem along the lines of its initial state (no expert system/ software for restaurant recommendations to Minervans), the goal state (create a curated expert system), and the scope (Minervan community, specifically in Buenos Aires Gorriti res) to ensure the proposed solutions addressed the right problem. We identify obstacles (varying time zones, language barrier, limited expertise) that might lead to ineffective approaches and mitigate them for a robust solution. We justify the utility of the solution by the potential threat of leaving the problem unsolved, i.e, hungry, frustrated Minervans in BA, concluding that this is an effective characterization that leads to an effective approach to the problem—an expert system.

#evidencebased - We utilize this HC in the use of relevant and persuasive evidence to support arguments (selecting credible online resources, testing the expert system in real-life). For the former argument, we provide a detailed google sheet with the data and describe the entire process in detail, mentioning the appropriate tools e.g. Google Maps, Rappi, and PepidosYa. For the latter, we provide visual evidence in the form of a photograph to prove that we went there.

#sourcequality - We use this HC during the task of finding restaurant information, to ensure that data collection was only from reputable sources that offer currency, accuracy, and relevance. Hence, in our case relying on Google Maps, Rappi, and PepiposYa for rating and distance measurements is a good strategy as Google is reputable for accuracy, and the duo is local, therefore they offer relevance. We relied on the concierge for the currency of information as they give Mineran's recommendations regularly, and would be in the know of what is 'hip'—a factor relevant to our problem case.

#modeling - We utilize this HC for the purpose of testing that the expert system works appropriately without any bugs, i.e, the expected results are observed from a given set of input parameters. We designed four test cases (no match, exactly 1 match, and more than 1 match, duplicate responses) that would cover the ground for basic tests, offering robust test models. We justify the accuracy of the models in the inference of how the system would behave with the change of input they provide, asserting that the expected results were observed. E.g, the last test checks the scenario where meat leads to 2 options, and the two actual options are accurately arrived at.

Appendix B: LOs

#aicoding - We apply this LO at the level of effectively implementing AI algorithms in the form of the expert system in Prolog that addresses the relevant problem we identified in section 1—a curated restaurant recommendation AI as an expert system. We push our application further by implementing the extensions (menu-based responses, and natural language queries) in a python interface to improve the user experience of the expert system application, ensuring that all implementation was computationally efficient and well-commented.

#ailogic - We apply this LO in the use of prolog to code and design the restaurant expert system by accurately generating appropriate predicates and propositions. We portray the soundness of this design by using color-coded logic-flow visualizations with clear, detailed steps that show which values of the askables lead to what final result.

Appendix C: Figures

Figure 2. Tree diagram of the expert system.

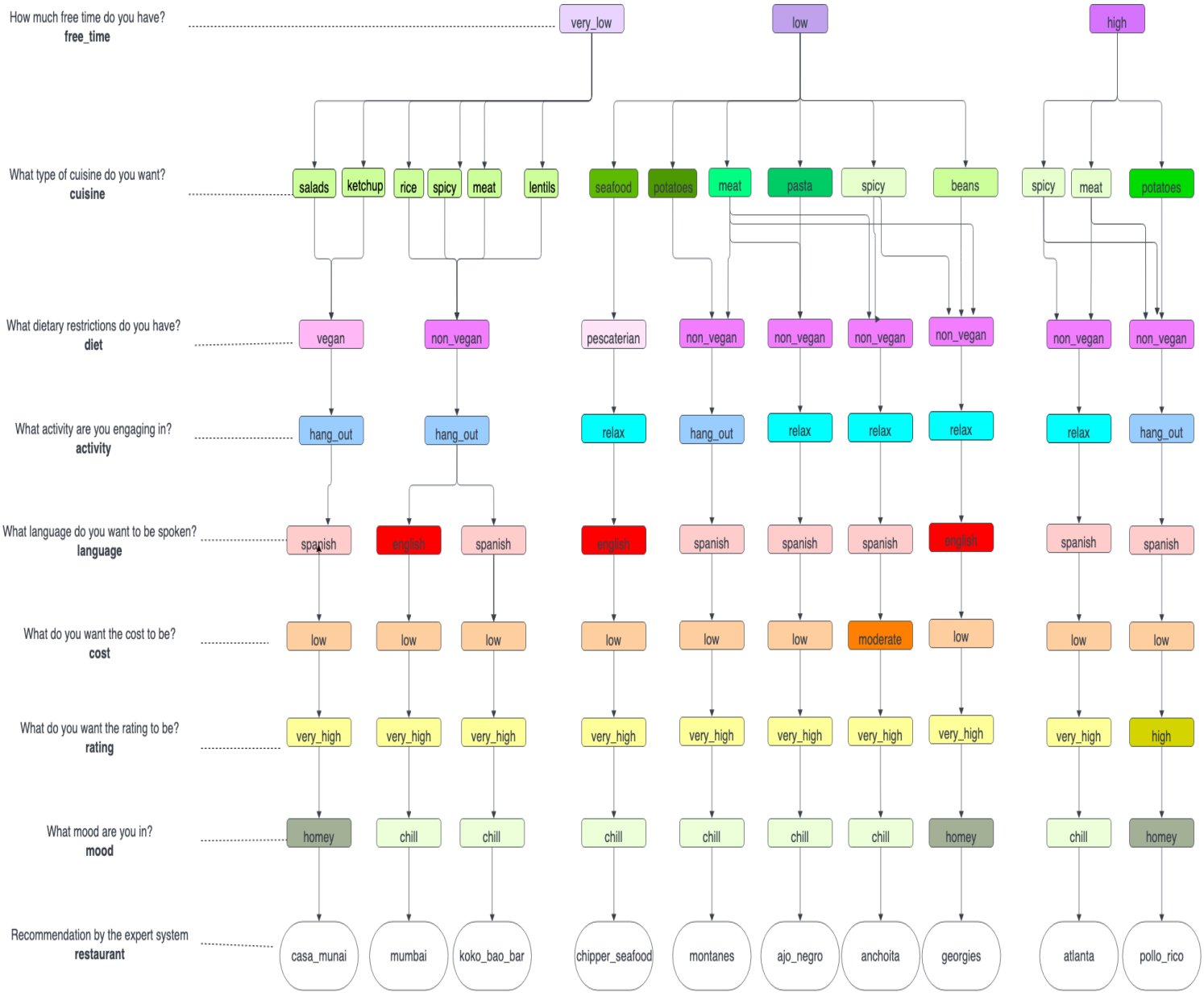

Figure 3. Group photo in front of Casa Munay restaurant.

Appendix D: Code

Forum Code Doc

```
In [ ]
!pip install pyswip

KB = """
%  Tell prolog that known/3 and multivalued/1 will be added later
:- dynamic known/3, multivalued/1.

% Enter your KB below this line:
restaurant(casa_munay) :- distance(very_close), cuisine(western), diet(vegan), service_option(dine_in), language(spanish), cost(low),
rating(very_high), atmosphere(cosy).
restaurant(koko_bao_bar) :- distance(very_close), cuisine(asian), diet(non_vegan), service_option(dine_in), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(mumbai) :- distance(very_close), cuisine(indian), diet(non_vegan), service_option(dine_in), language(english), cost(low),
rating(very_high), atmosphere(casual).
restaurant(ajo_negro) :- distance(close), cuisine(italian), diet(non_vegan), service_option(delivery), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(montanes) :- distance(close), cuisine(spanish), diet(non_vegan), service_option(dine_in), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(chipper_seafood) :- distance(close), cuisine(caribbean), diet(pescatarian), service_option(delivery), language(english), cost(low),
rating(very_high), atmosphere(casual).
restaurant(anchoita) :- distance(close), cuisine(argentinian), diet(non_vegan), service_option(delivery), language(spanish), cost(moderate),
rating(very_high), atmosphere(fine_dining).
restaurant(georgies) :- distance(close), cuisine(mexican), diet(non_vegan), service_option(delivery), language(english), cost(low),
rating(very_high), atmosphere(cosy).
restaurant(atlanta) :- distance(far), cuisine(argentinian), diet(non_vegan), service_option(delivery), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(pollo_rico) :- distance(far), cuisine(peruvian), diet(non_vegan), service_option(dine_in), language(spanish), cost(low), rating(high),
atmosphere(cosy).

distance(very_close) :- free_time(very_low).
distance(close) :- free_time(low).
distance(far) :- free_time(high).
distance(very_high) :- free_time(very_high).

cuisine(western) :- ingredients(salads).
```

```
% If not multivalued, and already known to be something else, don't ask again for a different value.
ask(A, V):-
\+multivalued(A),
known(yes, A, V2),
V \== V2,
!, fail.

ask(A, V):-
read_py(A,V,Y), % get the answer
assertz(known(Y, A, V)), % remember it
Y == yes.   % succeed or fail
"""
```

Run Code

```
Requirement already satisfied: pyswip in /opt/conda/lib/python3.8/site-packages (0.2.10)

[notice] A new release of pip available: 22.1.2 -> 22.3.1
[notice] To update, run: pip install --upgrade pip
```

+ New Code + New Text

```
# The code here will ask the user for input based on the askables. It will only ask the user where necessary.

# Import necessary packages
import tempfile
from pyswip.prolog import Prolog
from pyswip.easy import *

prolog = Prolog() # Global handle to interpreter

retractall = Functor("retractall")
known = Functor("known",3)

# Define foreign functions for getting user input and writing to the screen
def write_py(X):
    print(str(X))
    sys.stdout.flush()
    return True

def read_py(A,V,Y):
    if isinstance(Y, Variable):
        response = input(str(A) + " is " + str(V) + "? ")
        Y.unify(response)
        return True
    else:
        return False

write_py.arity = 1
read_py.arity = 3

registerForeign(read_py)
registerForeign(write_py)

# Create a temporary file with the KB in it
(FD, name) = tempfile.mkstemp(suffix='.pl', text = "True")
with os.fdopen(FD, "w") as text_file:
    text_file.write(KB)
prolog.consult(name) # open the KB for consulting
os.unlink(name) # Remove the temporary file

call(retractall(known))
restaurant = [s for s in prolog.query("restaurant(X).", maxresult=2)]

if restaurant:
    print("Your restaurant option(s): ")
    for i in range(len(restaurant)):
        print("{}) ".format(i+1) + (restaurant[i]['X']))
else:
    print("There are no restaurants in our database that match your description.")
```

Run Code

```
free_time is very_low?
free_time is low?
free_time is high?
There are no restaurants in our database that match your description.
```

+ New Code + New Text

Python 3 (7GB RAM) | Edit Run All Cells Kernel Ready | Restart | Interrupt

Heading 1

Extensions 1 & 2

We decided to leave the main part of the code above. We copy-pasted it and implemented the first two extensions below. We hope this makes it clearer to follow our work and process while grading.

+ New Code + New Text

In [1]

```
#!pip install pyswip

KB = """
%  Tell prolog that known/3 and multivalued/1 will be added later
:- dynamic known/3, multivalued/1.
:- discontiguous menuask/3.
:- discontiguous ask/2.

% Asking clauses
ask(A, V):-  known(yes, A, V), !.
ask(A, V):-  known(_, A, V), !, fail.
% If not multivalued, and already known to be something else, don't ask again for a different value.
ask(A, V):-  \+multivalued(A), known(yes, A, V2), V \== V2,  !, fail.
ask(A, V):- write(A:V), write('? '), read(Ans), assertz(known(Ans, A, V)),  Ans == yes.

menuask(Attr, Val, _) :- known(yes, Attr, Val), !.
menuask(Attr, Val, _) :- known(yes, Attr, V), V \== Val, !, fail.
menuask(Attr, Val, List) :-
    read_menu_py(Attr, Ans, List),
    check_val(Ans, Attr, Val, List),
    asserta(known(yes, Attr, Ans)),
    Ans == Val.

check_val(Ans, _, _, List) :- member(Ans, List), !.
check_val(Ans, Attr, Val, List) :-
     write('We do not have any options for '), write(Ans), nl,
    menuask(Attr, Val, List).
check_val(X, A, V, MenuList) :- write('You cannot select '), write(X), write(', please try something else'), nl, menuask(A, V, MenuList).

multivalued(distance).
multivalued(cuisine).
multivalued(diet).
multivalued(service_option).
multivalued(language).
multivalued(cost).
multivalued(rating).
multivalued(atmosphere).

% Enter your KB below this line:
restaurant(casa_munay) :- distance(very_close), cuisine(western), diet(vegan), service_option(dine_in), language(spanish), cost(low),
rating(very_high), atmosphere(cosy).
restaurant(koko_bao_bar) :- distance(very_close), cuisine(asian), diet(non_vegan), service_option(dine_in), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(mumbai) :- distance(very_close), cuisine(indian), diet(non_vegan), service_option(dine_in), language(english), cost(low),
rating(very_high), atmosphere(casual).
restaurant(ajo_negro) :- distance(close), cuisine(italian), diet(non_vegan), service_option(delivery), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(montanes) :- distance(close), cuisine(spanish), diet(non_vegan), service_option(dine_in), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(chipper_seafood) :- distance(close), cuisine(caribbean), diet(pescatarian), service_option(delivery), language(english), cost(low),
rating(very_high), atmosphere(casual).
restaurant(anchoita) :- distance(close), cuisine(argentinian), diet(non_vegan), service_option(delivery), language(spanish), cost(moderate),
rating(very_high), atmosphere(fine_dining).
restaurant(georgies) :- distance(close), cuisine(mexican), diet(non_vegan), service_option(delivery), language(english), cost(low),
rating(very_high), atmosphere(cosy).
restaurant(atlanta) :- distance(far), cuisine(argentinian), diet(non_vegan), service_option(delivery), language(spanish), cost(low),
rating(very_high), atmosphere(casual).
restaurant(pollo_rico) :- distance(far), cuisine(peruvian), diet(non_vegan), service_option(dine_in), language(spanish), cost(low), rating(high),
atmosphere(cosy).

% very_far distance missing
% fine_dining missing in atmosphere
% The code below implements the prompting to ask the user:
distance(X) :- menuask(distance, X, [very_close,close,far,very_far]).
cuisine(X) :- menuask(cuisine,X,[western,argentinian,asian,indian,italian,spanish,caribbean,mexican,peruvian]).
diet(X) :- menuask(diet, X,[vegan, non_vegan, pescatarian]).
service_option(X) :- menuask(service_option, X,[dine_in,delivery,takeaway]).
language(X) :- menuask(language, X,[english,spanish]).
cost(X) :- menuask(cost,X,[very_low,low,moderate,high,very_high]).
rating(X) :- menuask(rating,X,[very_low,low,moderate,high,very_high]).
atmosphere(X) :- menuask(atmosphere,X,[cosy,casual,fine_dining]).
"""
```

Python 3 (7GB RAM) | Edit Run All Cells Kernel Ready | Restart | Interrupt

In [1]

```python
# The code here will ask the user for input based on the askables. It will only ask the user where necessary.

# Import necessary packages
import tempfile
from pyswip.prolog import Prolog
from pyswip.easy import *

prolog = Prolog() # Global handle to interpreter

retractall = Functor("retractall")
known = Functor("known",3)

#Generate python UI/UX
translation = {
    "distance": "I want a restaurant that is",
    "cuisine": "I want to eat",
    "diet": "I have a dietary preference for",
    "service_option": "I want to",
    "language": "I want the staff to speak",
    "cost": "My budget for the restaurant is",
    "rating": "The restaurant's rating should be",
    "atmosphere": "I want an atmosphere that is",
}

def generate_question(category, lst):
    cg = " ".join(str(category).split("_"))

    if cg == "diet":
        return f"\nSelect an option for your {cg} preferences from {1}-{len(lst)}\n"

    if cg == "language":
        return f"\nSelect an option for preferenced {cg} from {1}-{len(lst)}\n"

    return f"\nSelect an option for the restaurant's {cg} from {1}-{len(lst)}\n"

def generate_option(category, strng):
    sentence = translation[category]
    option = " ".join(str(strng).split("_"))

    if category == "cuisine":
        return sentence + " " + option.capitalize()
    return sentence + " " + option

# Define foreign functions for getting user input and writing to the screen
def write_py(X):
    print(str(X))
    sys.stdout.flush()
    return True

def read_py(A,V,Y):
    if isinstance(Y, Variable):
        response = input(str(A) + " is " + str(V) + "? ")
        Y.unify(response)
        return True
    else:
        return False

def read_menu_py(A, Y, Menu):
    if isinstance(Y, Variable):
        print(generate_question(A, Menu))
        for i in range(len(Menu)):
            item = generate_option(str(A), Menu[i])
            if i == len(Menu):
                print(f"{i+1}. {item}\n")
            else:
                print(f"{i+1}. {item}")

        user_input = (input(f"\nYour choice: "))
        #process input - case number
        try:
            response = str(Menu[int(user_input)-1])
        except:
            response = str(user_input).lower()
        Y.unify(response)
        return True
    else:
        return False

write_py.arity = 1
read_py.arity = 3
read_menu_py.arity = 3
```

Python 3 (7GB RAM) | Edit Run All Cells Kernel Ready | Restart | Interrupt

```
registerForeign(read_py)
registerForeign(read_menu_py)
registerForeign(write_py)

# Create a temporary file with the KB in it
(FD, name) = tempfile.mkstemp(suffix='.pl', text = "True")
with os.fdopen(FD, "w") as text_file:
    text_file.write(KB)
prolog.consult(name) # open the KB for consulting
os.unlink(name) # Remove the temporary file

call(retractall(known))
restaurant = [s for s in prolog.query("restaurant(X).", maxresult=2)]

if restaurant:
    print("Your restaurant option(s): ")
    for i in range(len(restaurant)):
        print("{}) ".format(i+1) + (restaurant[i]['X']))
else:
    print("\nThere are no restaurants in our database that match your description.")
```

Running...

Out [13]

```
Select an option for the restaurant's distance

1. I want a restaurant that is very close
2. I want a restaurant that is close
3. I want a restaurant that is far
4. I want a restaurant that is very far
```

Your choice (1-4): Submit

+ New Code + New Text

Python 3 (7GB RAM) | Edit Run All Cells Kernel Busy | Restart | Interrupt