 University of Toronto - Computer Science - Image

Restaurant Recommendations: Planning the Program

At this point, you should:

- understand the problem, and
- be familiar with the data structures we'll use to represent the data from the file.

Next, we'll design the functions.

Quick Review:

We've created 3 dictionaries:

1. restaurant to rating (`dict of {str: int}`)
2. price range to restaurants (`dict of {str: list of str}`)
3. cuisine type to restaurants (`dict of {str: list of str}`)

Planning the Program

1. As with many programs, the first step is reading the file and building the data structures (in this case, building the three dictionaries).
2. Now we need to make a design decision: should we look up restaurants by price or by cuisine first? We choose to look up price first. (Choosing to look up restaurants by cuisine first would result in a different solution.) We'll make a list of the restaurants in the desired price range.
3. Next, we need to create a list of restaurants in the desired price range list that serve one of the desired cuisines.
4. Now that we have a list of restaurants in the desired price range that serve the desired cuisine type, we need to sort the restaurants based on their ratings and return the sorted list.

Design Tip

- Before programming, plan your program by breaking it down into subproblems. Take notes.
- Work through at least one example by hand and decide how to represent the data.
- Decompose the problem into subproblems so that you have more places in the program to test for correctness.

More Planning and Some Implementation

Step 1 - Build the Data Structures

Start by building the data structures we've discussed:

- `a dict of {restaurant name: rating %}`
- `a dict of {price: list of restaurant names}`
- `a dict of {cuisine: list of restaurant names}`

To do all of this in one function would be long, messy and hard to debug.Therefore, we create another function that takes a file and returns the three dictionaries as a tuple:

```
name_to_rating, price_to_names, cuisine_to_names = read_restaurant(file)
```

We can now create the header for this new function and create three accumulator variables. The function will accumulate the information from the file in the three dictionaries as it reads the file line-by-line:

```
def read_restaurants(file):
    """ (file) -> (dict of {str: int}, dict of {str: list of str}, dict of {str: list of str})

    Return a tuple of three dictionaries based on the information in file:

    - a dict of {restaurant name: rating%}
    - a dict of {price range: list of restaurant names}
    - a dict of {cuisine: list of restaurant names}
    """

    name_to_rating = {}
        price_to_name = {'$':[], '$$':[], '$$$':[], '$$$$':[]}
        cuisine_to_name = {}
```

Finishing this function is left as an exercise for you. We recommend you approach it as follows:

- Read the file line by line, and build up the `name_to_rating` dictionary.
- Only once that's working, go back and edit the function to build up the `price_to_name` dictionary as well.
- Next, add the code to build the `cuisine_to_name` dictionary.
- You should only read through the file once, so you'll need to add more statements in your for loop to build the 2nd and 3rd dictionaries.
- One thing that you may find a bit challenging is pulling apart the comma separated list of cuisines for the `cuisine_to_name` dictionary. You may want to create a helper function to aid with this task.

Step 2 - Filter by Price Range

We now have:

- the `price_to_name` dictionary, and
- the price range the user desires.

We can find the names of restaurants in that price range as follows:

```
names_matching_price = price_to_names[price]
```

Step 3 - Filter by Cuisine

We now need a new list of restaurants from the `names_matching_price` list that we just created that also serve the desired type of cuisine. We'll add a helper function `filter_by_cuisine` to return a list of all of the restaurants that are in the price range that serves one of the cuisines in the `cuisines_list`.

```
names_final = filter_by_cuisine(names_matching_price, cuisine_to_names, and cuisines_list)
```

Start by writing following the Function Design Recipe to write the function header:

```
def filter_by_cuisine(names_matching_price, cuisine_to_names, cuisines_list):
    """ (list of str, dict of {str: list of str}, list of str) -> list of str

    Return a list of the restaurants in names_matching_price that serve at
    least one of the cuisines in cuisines_list according to cuisine_to_names.

    >>> names = ['Queen St. Cafe', 'Dumplings R Us', 'Deep Fried Everything']
    >>> cuis = {'Canadian': ['Georgie Porgie'],
                'Pub Food': ['Georgie Porgie', 'Deep Fried Everything'],
                'Malaysian': ['Queen St. Cafe'],
                'Thai': ['Queen St. Cafe'],
                'Chinese: ['Dumplings R Us'],
                'Mexican': ['Mexican Grill']}
```

```
    >>> cuisines = ['Chinese', 'Thai']
    >>> filter_by_cuisine(names, cuis, cuisines)
    ['Queen St. Cafe', 'Dumplings R Us']
    """
```

In the example in the docstring, we filtered the `names_matching_price` list, looking for Thai (`'Queen St. Cafe'`) and Chinese (`'Dumplings R Us'`). We expect those two restaurants to be returned: `['Queen St. Cafe', 'Dumplings R Us']`.

For this example, we would start by looking for `'Queen St. Cafe'` in the `'Chinese'` food list. It's not there, so we now check the `'Thai'` food list. It is there, so we add `'Queen St. Cafe'` to our accumulator. We then move onto `'Dumplings R Us'` and then `'Deep Fried Everything'` repeating the same process.
Since `'Deep Fried Everything'` doesn't appear in either the Chinese or Thai lists, it is excluded from the results.

This function is left as an exercise for you to implement.

Step 4 - Sort and Return

We need to build a `list of list` in the right format, and sort by rating percentage. Again, we'll use a helper function:

```
result = build_rating_list(name_to_rating, names_final)
```

Again, start with the function header:

```
def build_rating_list(names_matching, name_to_rating):
    """ (list of str, dict of {str: int}) -> list of [int, str] list

    Return a list of [rating%, restaurant name], sorted by rating%.
    >>> name_to_rating = {'Georgie Porgie': 87,
                          'Queen St. Cafe': 82,
                          'Dumplings R Us': 71,
                          'Mexican Grill': 85,
                          'Deep Fried Everything': 52}
    >>> names = ['Queen St. Cafe', 'Dumplings R Us']
    >>> build_rating_list(name_to_rating, names)
    [[82, 'Queen St. Cafe'], [71, 'Dumplings R Us']]
    """
```

Final steps

Now, all that's left is to sort and return the list of recommended restaurants. The reason we put the rating first, followed by the string, is that when calling `sort()` on a `list of list`, it will use the first value of the inner list to determine the sort order of the items.

The rest of this function is being left for you to complete.

Jennifer Campbell • Paul Gries
University of Toronto
