**Notable Obstacles I Overcame:**

* One obstacle I had was trying to sort out which if statements were responsible for initiating which blocks of code; this became difficult when large sequences of if statements were repeatedly nested underneath each other. The challenge was overcome by paying particular attention to the indentation of each line, as well as by placing comments on multiple lines of code to more easily delineate the function of each if statement.
* Another obstacle was deciphering what certain error messages meant after writing large amounts of code at once and running it. After looking at the project 2 FAQ page, I noticed that the advice of writing and running each segment of the program separately was very helpful in increasing the efficiency with which I could detect both compile-time and logic errors.
* A final obstacle I faced was learning how to comment out a large block of code. Instead of placing the // character in front of each line, I eventually figured out that “command” + “/” on a mac could be used to comment out multiple lines at once.

**Possible Input Data to Test My Program:**

* In order to test my program, each possible circumstance that produces an associated meal cost should be imputed to see if it gives the correct meal cost. For example, a sample sequence of such user input might be:

BruinCard → Dinner → Student → Yes

in which case the output of the program should read:

Thank you for eating at UCLA today. Your meal cost is $*12.25*.

* The above input sequence should be repeated for every combination of payment method, type of meal, type of diner, and state of residence so as to ensure that the program outputs the correct meal cost for each circumstance.
  + The complete list of all necessary test inputs is as follows:
    - Ticket → Breakfast
    - Ticket → Lunch
    - Ticket → Dinner
    - BruinCard → Breakfast → Staff
    - BruinCard → Lunch → Staff
    - BruinCard → Dinner → Staff
    - BruinCard → Breakfast → Student → Yes
    - BruinCard → Breakfast → Student → No
    - BruinCard → Lunch → Student → Yes
    - BruinCard → Lunch → Student → No
    - BruinCard → Dinner → Student → Yes
    - BruinCard → Dinner → Student → No
* In addition, each error message should be tested by introducing unallowed input values or empty strings. Each error message should correspond to the question being asked, so, for example, an erroneous answer to

Ticket or BruinCard?

should produce the output

The kind value must either be Ticket or BruinCard.

Program ended with exit code: 1.

The correct corresponding error messages should be outputted in response to every erroneous answer or empty string inputted after a question prompt so as to ensure that the program reacts correctly to user input errors.

* The program I created handles all of these test cases correctly.