Testing
Nutrient+
Nutrient+ Team
12/2/19

Unit Testing

- Andi Zhao:
  - Tested the calculate function and populate function in ViewController.swift on the main page using print statements.
  - Fetched and tested data from CoreData to ensure all data is passed correctly.
  - Did some UI black box testing from the profile setup page to the edit targets page and the search food page to make sure the product does not crash with null pointer exceptions.
  - Tested for some sql statement testing to make sure database works

- Robert Sato:
  - I tested the SQLiteDatabase.swift file. This file gives basic CRUD functionality.
  - Testing was done manually checking the data with a printNutrTable() function.
  - After calling functions such as addNutr(), updateWeight(), and deleteTable(), the expected output is compared to the output printed from the printNutrTable() function.
  - For all of these database functions, “Protein” and “Energy” were used as the test cases for String inputs. -1, 0, and 1 were used as the test cases for Double inputs (progress, weight, target).
    - The -1 input for progress, weight and target does not throw errors although it should.

- Huanlei Wu:
  - The view controller corresponding to *Startup.swift* and the 3 pages before it should only display once at the very beginning. In order to ascertain this trait, I tested it by adding a line of code that allows me to revisit the startup view controllers. For example, if I deleted the persistent variable and the 3 view controllers didn’t show up again, this means that the variable wasn’t saving properly.
  - I tested *SaveUserInfo.swift* by using print statements to check if the user’s inputs were being saved properly into Core Data. I did this by first storing the inputs into the database and then calling the database to print out the individual attributes of

the table. If the information I inputted wasn't the same as the Core Data's information, this meant that my function in *SaveUserInfo.swift* wasn't saving properly.

    - I tested *EditProfile.swift* by doing the same thing I did with *SaveUserInfo.swift*. Testing whether the old Core Data (the user input during the startup) information was displayed in the text field was done through the UI.

- Victor:
    - I tested the nutrient recommendation algorithm which queried a preloaded bundled SQLite database to return a series of possible recommendations. Each query required a respective nutrients progress and target, and returned 30 fields per food, all of which would be used as the input of the recommendation function. Utilizing the black box testing concept, the inputs computed into a total value which took into account their respective weights. A series of print statements were also used to ensure that no illegal values were created or used. Further, mock values were also used to test the algorithm.
    - I tested the food adding functionality by simply running and studying the food API. As the functionality required two API requests, I needed to ensure that the result of the first request was compatible with the second. As such, I used print statements to check whether or not the results were legal or not. Finally, as the database returned JSON objects were not entirely consistent, I had to test this by running a large number of requests to account for outliers.

- Max:
    - I tested the initial database implementation in the ViewController.swift file by using print statements to check the create and delete methods of the CRUD functionality. While we eventually moved away from using the database implementation within the ViewController, testing done influenced the later implementation in the SQLiteDatabase.swift file.
    - I assisted Robert in testing the initial SQLiteDatabase.swift file, as we were pair programming at the time. I only tested the initial CRUD functions with him, before he implemented the database across the application to account for non-persistent data.