COSC 101 Homework 8: Fall 2021

Due date: **Thursday, November 11, 5:00pm**

Introduction

According to data from Feeding America, 9.9% of people in Madison county (which is where Hamilton is located) faced food insecurity in 2019, and Feeding America projected the rate increased to 12.5% in 2020 due to the pandemic. The Hamilton Food Cupboard is a critical community resource for individuals facing food insecurity. They operate on a client choice food pantry model, in which individuals can choose what products they receive, just like shopping at a grocery store.

To make the Hamilton Food Cupboard even more convenient and welcoming to community members needing food assistance, the Food Cupboard has (fictitiously) decided to offer food pick-up/delivery service similar to companies like Instacart. Your task is to write a program that clients can use to select which foods they would like to receive from the Food Cupboard.

Please write your code in the template file `hw8_food.py` included with this homework. The file includes an empty `main` function and some global variables that have been pre-populated for your program to use.

Required functionality

Welcome banner

Your program should begin by displaying the welcome banner included in the `hw8_food.py` file:

```
                          WELCOME TO THE HAMILTON

  ______              _    _____            _                         _ 
 |  ____|            | |  / ____|          | |                       | |
 | |__ ___   ___   __| | | |    _   _ _ __ | |__   ___   __ _ _ __ __| |
 |  __/ _ \ / _ \ / _` | | |   | | | | '_ \| '_ \ / _ \ / _` | '__/ _` |
 | | | (_) | (_) | (_| | | |___| |_| | |_) | |_) | (_) | (_| | | | (_| |
 |_|  \___/ \___/ \__,_|  \_____\__,_| .__/|_.__/ \___/ \__,_|_|  \__,_|
                                     | |                                
                                     |_|                                
```

Select a category

Next, the program should **display the categories of food** from which a user can select. The categories should be **displayed in the order they occur in the `categories` list** included in `hw8_food.py` and should be **numbered starting at 1**. After listing the categories, the program should **ask the user: `Which category do you want?`**

```
Categories:
1. Fruit
2. Vegetable
3. Grain
4. Dairy
Which category do you want?
```

If the user enters **a non-numeric value or a number that is not one of the displayed options**, then the program should **print the message `Enter a number between 1 and _` (replacing _ with the largest possible number) and again ask the user `Which category do you want?`**. The process should repeat until the user enters a valid choice. For example:

```
Which category do you want? F
Enter a number between 1 and 4
Which category do you want? Fruit
```

```
Enter a number between 1 and 4
Which category do you want? 10
Enter a number between 1 and 4
Which category do you want? 1
```

Select a food

Next, the program should **display the available foods in the selected category**. The `foods` list included in the `hw8_food.py` includes foods across all categories. A food's category can be determined by looking at the corresponding index in the `categories` list. For example, `foods[1]` is `"Cauliflower"` and `categories[1]` is `"Vegetable"`. Similar to above, the foods should be **displayed in the order they occur in the `foods` list** and should be **numbered starting at 1**. After listing the foods, the program should **ask the user: `Which food do you want?`**

For example, if the user choose the "Fruit" category, the program would output the following:

```
Foods:
1. Bananas
2. Mangoes
Which food do you want?
```

As above, if the user enters **a non-numeric value or a number that is not one of the displayed options**, then the program should **print the message `Enter a number between 1 and _` (replacing _ with the largest possible number) and again ask the user `Which food do you want?`**

Desired units

Next, the program should display the client's remaining credits for selecting food. In the client choice food pantry model, clients are normally given a credit budget based on a number of factors (e.g., household size), and each item is assigned a "cost" in credits. For simplicity, your program will assume a client **starts with 10 credits** and **each unit of a food costs one credit** regardless of the category or specific food.

The program should **ask the user: `How many units do you want?`** If the user enters **a non-numeric value or a number greater than their remaining credits**, then the program should **print the message `Enter a number between 1 and _` (replacing _ with the number of remaining credits) and again ask the user `How many units do you want?`**

For example:

```
Remaining credits: 10
How many units do you want? 12
Enter a number between 1 and 10
How many units do you want? five
Enter a number between 1 and 10
How many units do you want? 5
```

The number of units should be deducted from the client's remaining credits. If the client still has remaining credits, then the user should again be asked to select a category, select a food, and enter the desired number of units.

Display summary

After the client has used all of their credits, the program should display the foods they requested and the number of each item. For example:

```
Foods requested:
* Bananas (x8)
* Milk (x2)
```

The items must be **displayed in the order they were selected** by the user.

Note that a user may request some units of an item then request more units of the same item, but the item should only appear once in the order summary with the total amount requested.

For example:

```
Categories:
1. Fruit
2. Vegetable
3. Grain
4. Dairy
Which category do you want? 3
Foods:
1. Wheat bread
2. Rice
Which food do you want? 2
Remaining credits: 10
How many units do you want? 6

Categories:
1. Fruit
2. Vegetable
3. Grain
4. Dairy
Which category do you want? 3
Foods:
1. Wheat bread
2. Rice
Which food do you want? 2
Remaining credits: 4
How many units do you want? 4

Foods requested:
* Rice (x10)
```

Program structure

Past homeworks provided a (recommended) structure for your program. For this homework, **you** must decide how to structure your program. Your program **must contain multiple functions**. Also, your program **should not contain repeated code**—if you find yourself copying and pasting (nearly) the same code, should create a function that can be called from multiple points in your program. For all functions you write you should try to make sure that:

- any function you write does *one* thing (i.e., one specific task);
- any function function you write just takes parameters and returns a return value (or return values), but does *not* modify any global state or global variables;
- any function you write is fairly *short*

Write a little bit of code at a time (one function at most, but maybe just a couple lines in a function) and test as you go. Writing most all the code you think you need and testing afterward is a strategy that will often lead to frustration. Do a little bit at a time.

Example output

The following pages demonstrate what a correctly working program should look like. Study these examples closely!

- Trace 1
- Trace 2

Grading

Your assignment will be graded on two criteria:

1. Correctness (75%): your game must be implemented correctly. Refer to required functionality and example output above to help ensure that your program works the way it should.
2. Program design and style (25%): program design is particularly important for this assignment. Organize your functions so that each one is relatively short does *one* thing and takes few arguments. Think about how to break down the problem of "playing wheel of fortune" into smaller subtasks. And as usual:
   - Functions should have meaningful docstrings
   - Variable names should be meaningful
   - Programs should contain at least a few descriptive comments. Do *not* comment every line of code with low level explanations of what each line does. Focus on high level ideas.
   - All code should be structured so that the logic is clear and easy to follow.
   - There should be *no global variables* except for the three included in the provided code: `banner`, `categories`, and `foods`. Any other information needed inside a function should be passed as parameters; anything done by a function that needs to be used by the caller of the function should be returned from the function.

Testing

There are no automated tests for this homework. You'll need to run your code and test it in a variety of ways to ensure that it behaves according to the specifications above. You are strongly encouraged to write `doctest` tests for your functions, but you are not required to do so.

Submission

Please upload only your `food.py` file.

Challenge problems

There are two options for challenge problems for this homework, as described below.

Challenge problem option 1: Only display foods not previously selected

Modify the program such that it only lists foods a user has **not** already selected. For example, if a user's first selection is `Bananas`, then the next time the Fruit options are displayed, only `Mangoes` should be displayed. If the user has already selected all foods in a category, then the category should no longer be displayed. For example, if the user's first choice is `Bananas` and their second choice is `Mangoes`, then the `Fruit` category should not be displayed for the user's third choice and beyond.

Challenge program option 2: Group order summary by category

Modify the program such that summary groups foods by category. For example:

```
Foods requested:
* Fruits
  * Bananas (x2)
  * Mangoes (x1)
* Grains
  * Rice (x2)
* Dairy
  * Yogurt (x3)
  * Milk (x2)
```