Assignment: Functions

Learning Objectives:

- Understand the purpose of functions.
- Practice implementing and using functions.
- Ensure accuracy when implementing logic for financial transactions.
- Reflect on predatory practices built into online shopping platforms.

Part 0: Getting Started

Download the following files:

- **`products.csv`**: contains shopping items
- **`shopping_simulator.py`**: functions used in run_simulator.py
- **`run_simulator.py`**: the shopping simulator

You will be implementing the functions declared in **`run_simulator.py`**. These functions are then used in the file **`run_simulator.py`** to power the interactive shopping simulator! Notice how the functions are reused throughout the code. **Do not** change the code in **`run_simulator.py`** (or if you want to play around with it, change it back after). You can think of the simulator as a very simple version of your favorite online shopping website.

The first function, **`read_items`**, in **`run_simulator.py`** is implemented for you. Use the provided docstrings to help you understand each function. The following instructions will give you some guidance as well.

Throughout this assignment, there will be reflection questions (indicated in italics) that you should answer in a file named **`reflection.txt`**. Include this in your submission. When completing reflection questions, feel free to do some of your own research!

Part 1: Purchase

The first function you should implement is **`apply_purchase`**:

Function Name	Description

`apply_purchase`	Subtract the cost of an item from the user's current balance. Parameters, cost and balance, are both floats. The function returns the updated balance after subtraction.

You may find it helpful to look at `run_simulator.py` and see how the function is being used.

Reflection: When writing code that handles money, programmers have a technical responsibility to be precise. What would happen if the `apply_purchase` function was wrong? How could you test it to make sure it works? How are financial calculations handled in the real-world? [**NOTE**: The last question is broad! Do some of your own research and provide a brief answer].

Part 2: Offer Products

Next, implement `offer_products`:

Function Name	Description
`offer_products`	Choose a random product from the .csv of products to offer the user. The function will take in the lists of products, categories, and prices from `read_items` and return a dictionary with the keys “name”, “category”, and “price” for the item chosen.

Reflection: In our implementation, we are choosing a product at random. How do you see products being advertised on online shopping platforms? Describe the logic or algorithm a company might use to offer products to online shoppers as they browse.

Part 3: Payment Plans

Next, you will implement two functions that will allow shoppers to use a payment plan when making purchases. If they do not have, say $100, available right now, they can split this charge up and pay it over time. Of course, there will be a small fee applied, so really the item costs, say $105, after it payments are complete.

Function Name	Description
`create_payment_plans`	This function creates a new payment plan and adds it to the list of all the users payment plans. Focus on creating the new plan first, the requirements include: (1) a total of 3 payments and (2) a 10% surcharge on the original product price. For example, if a product is $300, the new price is $330. This is $110 for 3 payments. Store the details of the payment plan in a dictionary containing the keys `num_payments` and `payment_amount`. Then append this dictionary to a list of all payment plans, which is a parameter value. The function will return this list of dictionaries.
`apply_payment_plans`	This function applies all active payment plans to the shoppers current balance and updates the list of payment plans. Here are two explicit steps: 1. Subtract payment amount from the user's balance. 2. Decrement the number of payments. Say I have 2 plans with 1 payment left, each of $20. This function would subtract $40 from my balance and then decrement the number of payments. Since the plan reached 0 payments, it is removed from the list. The payment plan list and balance are given as parameters. The function will return a tuple of the new balance and updated list of payment plans. You may want to use apply_purchase within this function to subtract payment amounts from the users balance. Remember that `payment_plans` is a list of dictionaries. What if the user has multiple payment plans?

Reflection: Have you seen companies offer payment plans before? Why do they do this?

Part 4: Discounts

The last function you will implement is `offer_discount`:

Function Name	Description
`offer_discount`	Randomly apply a 5-15% discount 50% of the time. The function takes in price as a parameter and returns a new price with the

	discount applied.

Reflection: When might companies offer a discount?

Part 5: Simulation

Now that you've implemented all the functions, try out the simulator! Feel free to change the data and simulator logic, just be sure to **turn in the original simulator file**.

When you're done playing around, answer the following reflection questions:

1. The decisions we make about our code change depending on what our goals are. How would you design each function if you were a company trying to maximize profits? What about if you were a consumer?
2. In your own words, define what it means to be "technically responsible" and "socially responsible" as a programmer.
3. Oftentimes, design decisions are not made by engineers. They are simply creating the systems that function how companies or organizations want them to. How does this make you feel? Can you think of examples of this? Where do programmers have freedom, where do they not? How might a programmer handle being asked to create something that goes against their values? [**NOTE**: There are many prompting questions here, answer freely without feeling constrained to answering each question exactly. Have something else to share, please do!]

Grading Rubric

Criteria	Proficient	Exemplary
Logic	● Calculations within functions are done correctly. ● The simulator works with little to no errors (i.e. errors are not noticeable or don't impede usage). ● Passes ≥10 of the Pytests.	● Meets all the Proficient requirements. ● The simulator works with no errors. ● Passes all of the Pytests.
Style	● Includes in-line comments. ● Variable names are descriptive. ● A Pylint score between 7.0 - 8.0.	● Meets all the Proficient requirements. ● Comments are descriptive, help improve readability, but aren't redundant. ● Pylint score of ≥ 8.0
Reflection	● Answers all reflection questions.	● Meets all the Proficient requirements. ● Answers to reflection questions are thoughtful. This includes using one or more of the following: ○ Real-world examples ○ Personal connections ○ Citing an article they read