Unit Testing of the checkoutAndPayment and update\_users\_json Functions

# checkoutAndPayment:

Objective of the checkoutAndPayment Function

The checkoutAndPayment function is designed to manage the checkout process of an online shopping system. It handles user input, updates the shopping cart, and performs the final checkout action including payment processing and updating user data.

Input Domain for checkoutAndPayment

The checkoutAndPayment function takes as input a dictionary containing user information (username and wallet amount). The valid input type is a dictionary with the specified keys. The input domain can thus be divided into two equivalence classes (EC):

1. Invalid input formats (non-dictionary, missing keys, incorrect types).
2. Valid dictionary format with correct keys and types.

Coverage Criteria for checkoutAndPayment

For each EC, we need to test at least one input to ensure the function behaves as expected.

Test Requirements for checkoutAndPayment

* R1. If input is in EC1 (Invalid formats), checkoutAndPayment should raise a TypeError.
* R2. If input is in EC2 (Valid format), checkoutAndPayment should correctly handle the checkout process.

Corresponding Test Cases for checkoutAndPayment

* TC1. Input: "invalid\_string" (Invalid format)
* TC2. Input: {"username": "testuser", "wallet": 100} (Valid format)

Fixtures and Mocking for checkoutAndPayment

For checkoutAndPayment, mocking user input and product data is essential. This includes mocking the responses for product selection, cart checking, and the logout process. This simulation allows for testing various user interaction scenarios within the checkout process without relying on real user inputs or actual product data.

update\_users\_json:

Objective of the update\_users\_json Function

The update\_users\_json function is responsible for updating the wallet amount of users in a JSON file. It handles adding new users and updating existing ones.

Input Domain for update\_users\_json

The update\_users\_json function takes as input a username, new wallet amount, and an optional file path. The input domain is divided into:

1. Valid user details with an existing username.
2. Valid user details with a new username.
3. Invalid inputs (non-string username, non-numeric wallet amount, invalid file path).

Coverage Criteria for update\_users\_json

For each EC, at least one input should be tested to validate function behavior.

Test Requirements for update\_users\_json

* R1. If input is in EC1 (Existing user), update\_users\_json should update the user's wallet in the JSON file.
* R2. If input is in EC2 (New user), update\_users\_json should add the new user to the JSON file.
* R3. If input is in EC3 (Invalid inputs), update\_users\_json should handle exceptions appropriately.

Corresponding Test Cases for update\_users\_json

* TC1. Input: ("user1", 150) (Existing user)
* TC2. Input: ("new\_user", 300) (New user)
* TC3. Input: ("user1", "invalid\_wallet") (Invalid wallet amount)

Fixtures and Mocking for update\_users\_json

For update\_users\_json, mocking file operations is crucial. This includes mocking the file read and write operations to prevent actual changes to the JSON file during testing. This approach allows for testing the function's ability to handle file updates, new user additions, and exception scenarios in a controlled environment without affecting real data files.