SHREYA GHOLASE
23070521145
B2

Final Task Practical 5 Cart Total Calculator

Problem Statement

You are building a **shopping cart calculator**. Each cart item is represented as an **object** with: { id, name, price, qty, category }

The program must calculate:

1. **Subtotal** (sum of all items `price × qty`).
2. **Discounts**:
   If quantity ≥ 3, apply **5% discount** on that item.
   If category = `"stationery"` and subtotal of stationery items > 200, apply **10% category discount**.
3. **Final Total** after discounts.

Steps

1. **Create the cart**: An array of objects (3–5 products, different categories).
2. **Use `map()`** to calculate each item's total (`price × qty`).
3. **Apply item-level discount** using `map()` or inside `reduce()`.
4. **Use `filter()`** to extract stationery items and calculate their subtotal.
5. **Apply category-level discount** if applicable.
6. **Use `reduce()`** to calculate overall subtotal and final total.
7. **Use `forEach()`** to print a formatted receipt.

Sample Input (Cart Example)

```
[
  { id: 1, name: "Pen", price: 20, qty: 2, category: "stationery" },
  { id: 2, name: "Mug", price: 150, qty: 1, category: "kitchen" },
  { id: 3, name: "Notebook", price: 80, qty: 3, category: "stationery" }
]
```

Expected Output (Approximate Format)

Item: Pen (x2) = 40
Item: Mug (x1) = 150
Item: Notebook (x3) = 240 → discount applied

Subtotal: 430
Item Discounts: 12
Stationery Discount: 26.8
Final Total: 391.2

Reminders

- Don't forget **quantity × price** when calculating totals.
- Discounts must be **subtracted after subtotal** is found.
- Use `reduce()` effectively to **accumulate sums**.
- Break the problem into **small reusable functions**.
- Format money properly (e.g., two decimal places).

Note: This is an **implementation lab**. Follow the hints carefully and **write the full program yourself**.

Detailed Steps:

Part A – Setup the Cart

1. Create an array named `cart`.
   Each element should be an object with properties:
   `{ id, name, price, qty, category }`.
   Example categories: `stationery`, `kitchen`, `electronics`.

Write your cart array below:

```
let cart = [
  _____,
  _____,
  _____
];
```

Checkpoint: At least **3–4 items** in your cart.

Part B – Subtotal per Item

1. Use `map()` to calculate the **subtotal** for each item (`price × qty`).
2. Print each item's subtotal.

Formula hint: `item.price * item.qty`

```
let itemTotals = cart.map(item => _____ );
console.log(itemTotals);
```

Checkpoint: Does your array show the correct subtotal values?

Part C – Item-Level Discount

Rule: If `qty ≥ 3`, apply a **5% discount** on that item.

Hint: Use a **conditional (`if` / ternary)** inside your calculation.

Fill in logic:

```
let discountedItemTotals = cart.map(item => {
  let subtotal = _____;
  if( _____ ) {
    subtotal = subtotal - (subtotal * 0.05);
  }
  return subtotal;
});
```

Checkpoint: Test with an item having `qty = 3` or more.

Part D – Category-Level Discount

Rule: If total **stationery subtotal > 200**, apply **10% off stationery subtotal**.

1. Use `filter()` to extract stationery items.
2. Use `reduce()` to sum their total.
3. If the total > 200, calculate discount.

Fill-in:

```
let stationeryItems = cart._____(item => item.category === "stationery"); let
stationeryTotal = stationeryItems._____( (sum, item) => sum + (item.price * item.qty),
0);

if( _____ ) {
  let stationeryDiscount = stationeryTotal * 0.10;
  console.log("Stationery Discount:", stationeryDiscount);
}
```

Checkpoint: Try with a stationery-heavy cart.

Part E – Final Total

1. Use `reduce()` to get the **cart subtotal**.
2. Subtract **item-level discounts**.
3. Subtract **category-level discount**.
4. Print the final total with two decimal places.

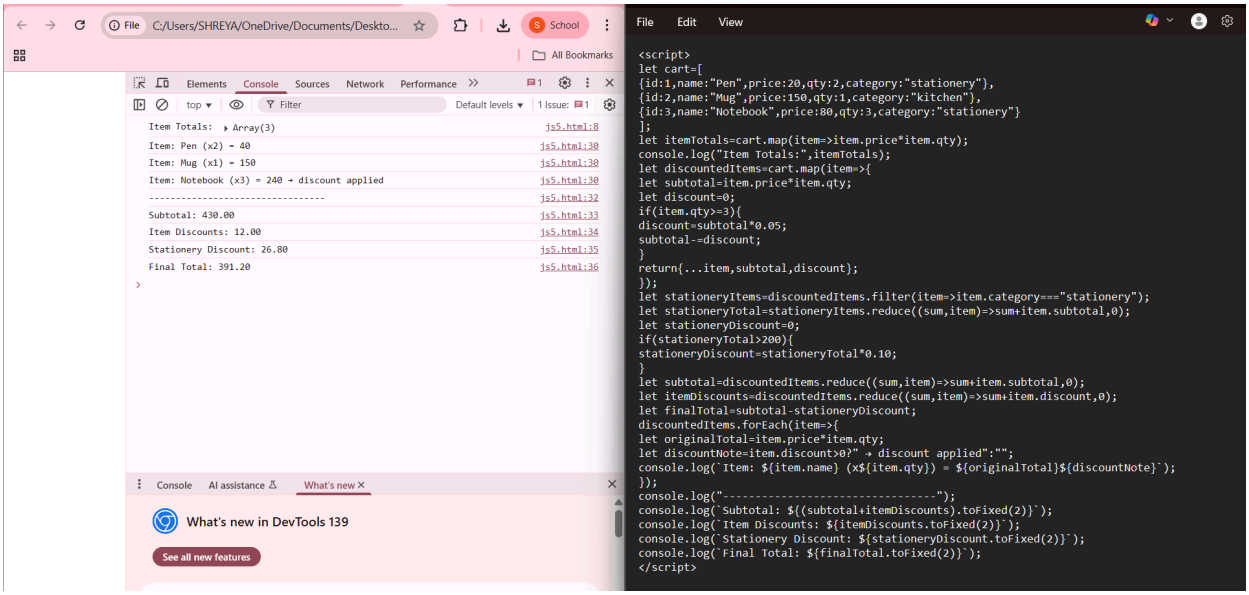
Reminder: `Number(value.toFixed(2))` can help format decimals.

Sample Input (For Testing)

```
[
  { id: 1, name: "Pen", price: 20, qty: 2, category: "stationery" },
  { id: 2, name: "Mug", price: 150, qty: 1, category: "kitchen" },
  { id: 3, name: "Notebook", price: 80, qty: 3, category: "stationery" }
]
```

Expected Output (Format Example)

```
Item: Pen (x2) = 40
Item: Mug (x1) = 150
Item: Notebook (x3) = 240 → discount applied
---------------------------------
Subtotal: 430
Item Discounts: 12
Stationery Discount: 26.8
Final Total: 391.2
```



Solve the following

- Which function (`map`, `filter`, `reduce`, `forEach`) was **hardest** to apply? Why?

  `reduce()` was the hardest to apply.
  Reason: Unlike `map()` or `filter()`, which work item by item in a straightforward way, `reduce()` requires keeping track of an accumulator and carefully deciding how to update it on each step.
  In our shopping cart, `reduce()` was used to calculate totals and discounts, which required combining multiple values into one. This makes it slightly more complex compared to `map()` (transform items) and `filter()` (select items).

- How could this program be extended (e.g., tax, coupons, shipping)?

  Taxes:

  - Add a fixed tax rate (e.g., 5% GST) on the subtotal after discounts.

2. **Coupons/Promo Codes:**

   - Add logic to apply coupon codes (e.g., "SAVE10" gives 10% off).

3. **Shipping Charges:**

- Add shipping fees (e.g., free shipping if total > ₹500, otherwise ₹50 charge).

4. **Membership Discounts:**
    - Extra discount for premium members (e.g., 5% off total).
5. **Dynamic Pricing:**
    - Prices could change depending on demand or time (like surge pricing).
6. **Multi-currency Support:**
    - Convert total into other currencies (USD, EUR, etc.) using exchange rates.

Important: This worksheet contains hints only. You must **write the complete working program** yourself.