Homework 1

Installing packages

```
!pip install langchain_google_genai
```

```
Requirement already satisfied: langchain_google_genai in /usr/local/lib/python3.12/dist-packages (4.2.0)
Requirement already satisfied: filetype<2.0.0,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (1.2.0)
Requirement already satisfied: google-genai<2.0.0,>=1.56.0 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (1.60.0)
Requirement already satisfied: langchain-core<2.0.0,>=1.2.5 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (1.2.7)
Requirement already satisfied: pydantic<3.0.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (2.12.3)
Requirement already satisfied: anyio<5.0.0,>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (4.12.1)
Requirement already satisfied: google-auth<3.0.0,>=2.47.0 in /usr/local/lib/python3.12/dist-packages (from google-auth[requests]<3.0.0,>=2.47.0->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (2.47.0)
Requirement already satisfied: httpx<1.0.0,>=0.28.1 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (0.28.1)
Requirement already satisfied: requests<3.0.0,>=2.28.1 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (2.32.4)
Requirement already satisfied: tenacity<9.2.0,>=8.2.3 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (9.1.2)
Requirement already satisfied: websockets<15.1.0,>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (15.0.1)
Requirement already satisfied: typing-extensions<5.0.0,>=4.11.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (4.15.0)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (1.9.0)
Requirement already satisfied: sniffio in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_genai) (1.3.1)
Requirement already satisfied: jsonpatch<2.0.0,>=1.33.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (1.33)
Requirement already satisfied: langsmith<1.0.0,>=0.3.45 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (0.6.4)
Requirement already satisfied: packaging<26.0.0,>=23.2.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (25.0)
Requirement already satisfied: pyyaml<7.0.0,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (6.0.3)
Requirement already satisfied: uuid-utils<1.0,>=0.12.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (0.13.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain_google_genai) (0.7.0)
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain_google_genai) (2.41.4)
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain_google_genai) (0.4.2)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-packages (from anyio<5.0.0,>=4.8.0->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (3.11)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google-auth[requests]<3.0.0,>=2.47.0->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (0.4.2)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google-auth[requests]<3.0.0,>=2.47.0->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (4.9.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (2026.1.4)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (0.16.0)
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.12/dist-packages (from jsonpatch<2.0.0,>=1.33.0->langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (3.0.0)
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (3.11.5)
Requirement already satisfied: requests-toolbelt>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (1.0.0)
Requirement already satisfied: zstandard>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core<2.0.0,>=1.2.5->langchain_google_genai) (0.25.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.28.1->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (3.4.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.28.1->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (2.5.0)
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3.0.0,>=2.47.0->google-auth[requests]<3.0.0,>=2.47.0->google-genai<2.0.0,>=1.56.0->langchain_google_genai) (0.6.2)
```

Setup your API key

To run the following cell, your API key must be stored it in a Colab Secret named `VERTEX_API_KEY`.

1. Look for the key icon on the left panel of your colab.
2. Under `Name`, create `VERTEX_API_KEY`.
3. Copy your key to `Value`.

```
from google.colab import userdata
GEMINI_VERTEX_API_KEY = userdata.get('VERTEX_API_KEY')
```

Downloading receipts.zip

The codes below download and unzip receipts.zip from Google Drive. receipts.zip contains all images from the Fusion folder on BlackBoard.

```
import gdown
file_id = "1oe2FZd3ZTO7nrDqjCafNvxic108oF8JF"
download_url = f"https://drive.google.com/uc?id={file_id}"
gdown.download(download_url, "receipts.zip", quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1oe2FZd3ZTO7nrDqjCafNvxic108oF8JF
To: /content/receipts.zip
100%|██████████| 1.61M/1.61M [00:00<00:00, 90.6MB/s]
'receipts.zip'
```

```
!unzip receipts.zip
```

```
Archive:  receipts.zip
replace receipt1.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt1.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace receipt2.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt2.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace receipt3.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt3.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace receipt4.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt4.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace receipt5.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt5.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace receipt6.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt6.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace receipt7.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace __MACOSX/._receipt7.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

1. Helper functions

We need two functions

- image_to_base64 convert your jpg image into Base64 encoded string (basically a sequence of 64 characters to make your image easily transfered via API)
- get_image_data_url takes your jpg image, converting them into base64 string and construct a suitable input for GEMINI api call.

```
import base64
import mimetypes

# Helper function to read and encode image
def image_to_base64(img_path):
    with open(img_path, "rb") as img_file:
        return base64.b64encode(img_file.read()).decode('utf-8')

# Helper function to encode local file to Base64 Data URL
def get_image_data_url(image_path):
    # Guess the mime type (e.g., image/png, image/jpeg) based on file extension
    mime_type, _ = mimetypes.guess_type(image_path)
    if mime_type is None:
        mime_type = "image/png" # Default fallback

    encoded_string = image_to_base64(image_path)

    # Construct the Data URL
    return f"data:{mime_type};base64,{encoded_string}"
```

```
from langchain_google_genai import ChatGoogleGenerativeAI
llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    api_key=userdata.get('VERTEX_API_KEY'), # Ensure this key is set in Colab secrets
    temperature=0,
    vertexai=True
)
```

Display jpg images. Alternatively, open the folder icon on the left pannel to see the images.

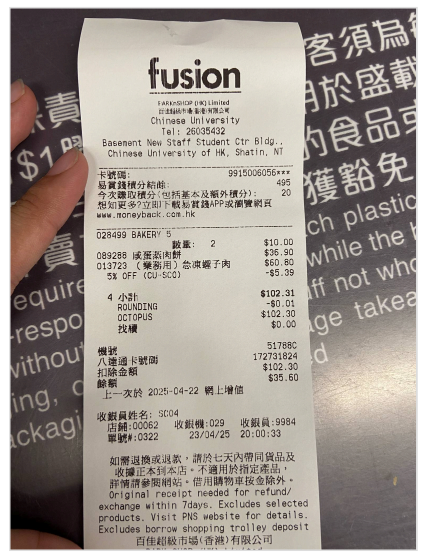
```
from IPython.display import HTML, display
import glob, os

image_paths = glob.glob("*.jpg")
image_paths.sort()
html_content = '<div style="display: flex; flex-wrap: wrap; gap: 20px;">'

for path in image_paths:
    b64 = image_to_base64(path)
    filename = os.path.basename(path) # Clean up path to show just the name

    # Create a vertical column for each image + text
    html_content += f'''
    <div style="display: flex; flex-direction: column; align-items: center;">
        <img src="data:image/jpeg;base64,{b64}" style="height: 300px; border: 1px solid #ddd; margin-bottom: 5px;"/>
        <span style="font-family: monospace; font-size: 14px;">{filename}</span>
    </div>
    '''

html_content += '</div>'

display(HTML(html_content))
```

receipt1.jpg

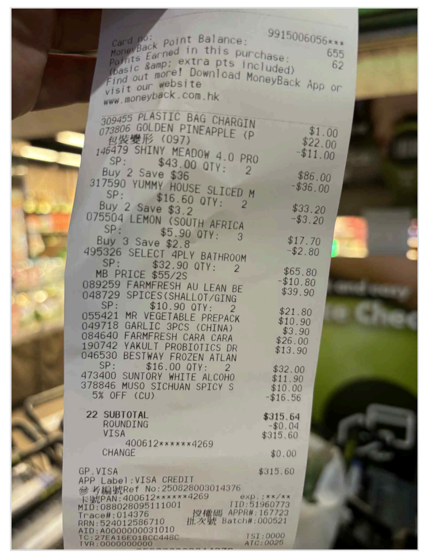
receipt2.jpg

receipt3.jpg

receipt4.jpg


receipt5.jpg

receipt6.jpg


receipt7.jpg

2. Image input to Gemini

Different from text, image needs to be converted into base64 encoded string and then formated into url before inputting to the language model. This is convenient for image-type input to be transfered through the API.

You can find out more appropriate format for image-type data in this Link

```
from langchain_core.prompts import ChatPromptTemplate
import json
import re

def clean_model_response(response_text):
    """Clean model response: remove ```json/``` tags, extra spaces/newlines, extract pure JSON string"""
    cleaned = re.sub(r'```json\s*|\s*```$', '', response_text.strip(), flags=re.MULTILINE)
    cleaned = re.sub(r'\s+', ' ', cleaned).strip()
    return cleaned

def calculate_original_price(step1_result):
    """Calculate original total price (without discount) based on extracted item details
    Calculation logic: Original total = Total paid amount + Sum of all discount amounts
    """
    discount_sum = 0.0

    for item in step1_result.get("item_details", []):
        discount = float(item.get("discount_amount", 0))
        if discount > 0:
            discount_sum += discount

    paid_total = float(step1_result.get("total_paid_amount", 0))

    final_original = paid_total + discount_sum
    return round(final_original, 2)

prompt_step1 = ChatPromptTemplate.from_messages([
    ("system", """You are a meticulous and accurate receipt price extraction expert. Output ONLY a valid JSON string (no other text, no code blocks, no markdown) and strictly comply wit
1. Image Analysis & Item Extraction:
   - Extract EVERY visible item/line item from the single receipt image (no omissions, even discount/promotion lines)
   - For each item, populate 4 mandatory fields (exact field names, case-sensitive):
     - "item_name": Exact name of the item (string; keep original spelling/case, e.g., "Milk", "5% OFF", "Tax"); fill "Unknown" if unrecognizable
     - "original_price": Undiscounted price of the item (numeric float; directly from the receipt's price column, BEFORE discount; 0 if it's a discount/promotion line with no base pric
     - "discount_amount": Exact amount reduced by discount (numeric float; convert negative discount values on receipt to positive, e.g., "-5.00" → 5.00; 0 if no discount for the item)
     - "total_paid_amount" (global field): Final total amount paid (numeric float; extract from "Total", "Amount Paid", "Grand Total" at the bottom of the receipt; 0 if unrecognizable)
2. JSON Structure (strictly follow, no extra fields/typos):
   {{
     "image_name": "EXACT filename of the receipt image (e.g., receipt1.jpg)",
     "item_details": [list of all extracted items (each with item_name/original_price/discount_amount)],
     "total_paid_amount": numeric float (final paid total)
   }}
3. Numeric Rules:
```

```
   - All amounts are pure numbers (float/int) with NO units (e.g., $, €), NO commas (e.g., 1,000.50 → 1000.50), NO currency symbols
   - Round all numeric values to 2 decimal places (standard currency format)
   - Fill 0 only if the field is completely unrecognizable (never leave fields empty/null)
4. Output Constraints:
   - Return ONLY the pure JSON string (no ```json, no comments, no explanations, no extra text)
   - Ensure JSON is valid (can be parsed by standard JSON parsers)
   - Match field names EXACTLY as specified (case-sensitive: e.g., "item_name" not "ItemName" or "itemname")"""),
    ("human", [
        {"type": "text", "text": "Extract all item details and total paid amount from the receipt image. Return ONLY a valid JSON string following the rules above (no code blocks, n
        {"type": "image_url", "image_url": {"url": "{image_url}"}}
    ])
])

chain_step1 = prompt_step1 | llm

final_all_results = []

image_data_urls = []
valid_image_paths = []
for img_path in image_paths:
    try:
        data_url = get_image_data_url(img_path)
        image_data_urls.append(data_url)
        valid_image_paths.append(img_path)
        print(f"✅ Image {os.path.basename(img_path)} converted to Data URL successfully")
    except Exception as e:
        print(f"❌ Failed to convert {os.path.basename(img_path)}, skipped: {str(e)}")
        continue

if len(image_data_urls) == 0:
    print("🚫 No valid images to process, skip model call!")
else:
    print(f"\n📢 Start processing {len(valid_image_paths)} images one by one...")
    for idx, (img_path, data_url) in enumerate(zip(valid_image_paths, image_data_urls)):
        img_name = os.path.basename(img_path)
        print(f"\n========= Processing image {idx+1}: {img_name} =========")

        print(f"🔹 Extracting item price details for {img_name}...")
        try:
            response_step1 = chain_step1.invoke({"image_url": data_url})
            step1_text = clean_model_response(response_step1.content)
            step1_result = json.loads(step1_text)
            step1_result["image_name"] = img_name
            print(f"✅ API call succeeded! Extracted {len(step1_result.get('item_details', []))} items")

            print(f"\n📜 {img_name} item bill details:")
            for i, item in enumerate(step1_result.get('item_details', []), 1):
                print(f"    {i}. {item.get('item_name', 'Unknown item')} → Original price: {item.get('original_price', 0)} | Discount: {item.get('discount_amount', 0)}")
        except json.JSONDecodeError as e:
            print(f"❌ Result parsing failed: {str(e)}, raw response: {response_step1.content[:200]}")
            continue
        except Exception as e:
            print(f"❌ API call failed: {str(e)}")
            continue

        original_price = calculate_original_price(step1_result)
        paid_price = round(float(step1_result.get('total_paid_amount', 0)), 2)
        print(f"\n💰 Calculation result → Total paid: {paid_price} | Original total (no discount): {original_price}")

        final_result = {
            "image_name": img_name,
            "total_paid_amount": paid_price,
            "original_total_amount": original_price
        }
        final_all_results.append(final_result)

if len(final_all_results) > 0:
    print("\n=================== Final Results Summary for All Images ===================")
    total_paid_all = 0.0
    total_original_all = 0.0

    for res in final_all_results:
        paid = res["total_paid_amount"]
        original = res["original_total_amount"]
        total_paid_all += paid
        total_original_all += original
        print(f"📄 {res['image_name']} → Total paid: {paid} | Original total (no discount): {original}")

    print(f"\n📊 All receipts summary:")
    print(f"✅ Total paid amount for all receipts: {round(total_paid_all, 2)}")
    print(f"✅ Total original amount (no discount) for all receipts: {round(total_original_all, 2)}")
else:
    print("\n🚫 No valid final results to summarize!")
```

```
  8. Buy 2 Save $0.8 → Original price: 0.0 | Discount: 0.8
```

```
   8. Buy 2 Save $0.8 → Original price: 0.0 | Discount: 0.8
   9. 059752 HIGH BRIX ORANGE FA → Original price: 32.0 | Discount: 0.0
   10. 089395 FARMFRESH BONELESS C → Original price: 34.9 | Discount: 0.0
   11. 053617 SUPREME BALANCE FZ N → Original price: 42.0 | Discount: 0.0
   12. 021965 POMELO (TAIWAN) → Original price: 9.5 | Discount: 0.0
   13. MB $200get 5%off → Original price: 0.0 | Discount: 10.52
   14. 5% OFF (CU-SCO) → Original price: 0.0 | Discount: 9.99
   15. ROUNDING → Original price: 0.0 | Discount: 0.09

💰 Calculation result → Total paid: 190.8 | Original total (no discount): 221.2

========== Processing image 7: receipt7.jpg ==========
🔹 Extracting item price details for receipt7.jpg...
✅ API call succeeded! Extracted 22 items

📜 receipt7.jpg item bill details:
   1. 309455 PLASTIC BAG CHARGIN → Original price: 1.0 | Discount: 0.0
   2. 073806 GOLDEN PINEAPPLE (P → Original price: 22.0 | Discount: 0.0
   3. 包裝變形(097) → Original price: 0.0 | Discount: 11.0
   4. 146479 SHINY MEADOW 4.0 PRO → Original price: 86.0 | Discount: 0.0
   5. Buy 2 Save $36 → Original price: 0.0 | Discount: 36.0
   6. 317590 YUMMY HOUSE SLICED M → Original price: 33.2 | Discount: 0.0
   7. Buy 2 Save $3.2 → Original price: 0.0 | Discount: 3.2
   8. 075504 LEMON (SOUTH AFRICA → Original price: 17.7 | Discount: 0.0
   9. Buy 3 Save $2.8 → Original price: 0.0 | Discount: 2.8
   10. 495326 SELECT 4PLY BATHROOM → Original price: 65.8 | Discount: 0.0
   11. MB PRICE $55/2S → Original price: 0.0 | Discount: 10.8
   12. 089259 FARMFRESH AU LEAN BE → Original price: 39.9 | Discount: 0.0
   13. 048729 SPICES(SHALLOT/GING → Original price: 21.8 | Discount: 0.0
   14. 055421 MR VEGETABLE PREPACK → Original price: 10.9 | Discount: 0.0
   15. 049718 GARLIC 3PCS (CHINA) → Original price: 3.9 | Discount: 0.0
   16. 084640 FARMFRESH CARA CARA → Original price: 26.0 | Discount: 0.0
   17. 190742 YAKULT PROBIOTICS DR → Original price: 13.9 | Discount: 0.0
   18. 046530 BESTWAY FROZEN ATLAN → Original price: 32.0 | Discount: 0.0
   19. 473400 SUNTORY WHITE ALCOHO → Original price: 11.9 | Discount: 0.0
   20. 378846 MUSO SICHUAN SPICY S → Original price: 10.0 | Discount: 0.0
   21. 5% OFF (CU) → Original price: 0.0 | Discount: 16.56
   22. ROUNDING → Original price: 0.0 | Discount: 0.04

💰 Calculation result → Total paid: 315.6 | Original total (no discount): 396.0

==================== Final Results Summary for All Images ====================
📄 receipt1.jpg → Total paid: 394.7 | Original total (no discount): 480.2
📄 receipt2.jpg → Total paid: 316.1 | Original total (no discount): 392.2
📄 receipt3.jpg → Total paid: 140.8 | Original total (no discount): 160.1
📄 receipt4.jpg → Total paid: 514.0 | Original total (no discount): 590.8
📄 receipt5.jpg → Total paid: 102.3 | Original total (no discount): 107.7
📄 receipt6.jpg → Total paid: 190.8 | Original total (no discount): 221.2
📄 receipt7.jpg → Total paid: 315.6 | Original total (no discount): 396.0

📊 All receipts summary:
✅ Total paid amount for all receipts: 1974.3
✅ Total original amount (no discount) for all receipts: 2348.2
```

3. Evaluation Code

- Make sure your LLM return a single float as the answer, stored in `query1_answer` and `query2_answer`
- Run the following code blocks: (1) If the blocks does not return any error, then your chain design is correct. Otherwise, please check your chain design.
- Do not modify `query_1_costs` and `query_2_costs`

```
def test_query(answer, ground_truth_costs):
    # Convert string to float if necessary
    if isinstance(answer, str):
        answer = float(answer)

    # Calculate the ground truth sum once for clarity
    expected_total = sum(ground_truth_costs)

    # Check if the answer is within +/- $2 of the expected total
    assert abs(answer - expected_total) <= 2
```

Run the following code block to evaluate query 1:

> How much money did I spend in total for these bills?

```
query_1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this
query1_answer = total_paid_all
test_query(query1_answer, query_1_costs)
```

Run the following code block to evaluate query 2:

> How much would I have had to pay without the discount?

````
query_2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this
query2_answer = total_original_all
test_query(query2_answer, query_2_costs)
````

````
sum([480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00])
````

````
2348.2
````

````
"""
Function Description:
Classify user queries into three categories:
1. Category 1: Query total paid amount for all receipts (corresponds to Query 1)
2. Category 2: Query total original amount (no discount) for all receipts (corresponds to Query 2)
3. Category 0: Other irrelevant questions
"""
from langchain_core.prompts import ChatPromptTemplate
import json
import re

prompt_classify = ChatPromptTemplate.from_messages([
        ("system", """You are a professional query classification assistant. Classify user queries about supermarket receipts strictly following these rules, and output only JSON string (no oth

### Classification Rules
Category 1 (Query total paid amount):
- Core intent: Ask for the total actual paid amount of all supermarket receipts
- Typical examples:
    - "How much money did I spend in total for these bills?"
    - "What is the total amount I actually paid for all these receipts?"
    - "Sum of total paid amount for all receipts?"

Category 2 (Query total original amount without discount):
- Core intent: Ask for the total amount to pay if there were no discounts for all supermarket receipts
- Typical examples:
    - "How much would I have had to pay without the discount?"
    - "What is the total original amount (no discount) for all receipts?"
    - "Total price without considering discounts for all bills?"

Category 0 (Other irrelevant questions):
- All questions not matching Category 1/2, including but not limited to:
    - Asking amount of a single receipt or price of a single item
    - Asking discount amount or discount rate
    - Irrelevant small talk, questions in other fields
    - Unrecognizable vague questions

### Output Requirements
1. Output format: Pure JSON string with 3 fields:
      - "query_text": Input user query text (preserved as-is)
      - "category": Classification result (string, options: "Query total paid amount", "Query total original amount (no discount)", "Other irrelevant questions")
      - "category_code": Classification code (integer, options: 1, 2, 0)
2. Currency unit and language (Chinese/English) do not affect classification, only focus on core intent
3. Return only JSON string, no extra explanations, comments, or code block tags
4. Unrecognizable questions are classified as Category 0 uniformly
"""),
        ("human", "Classify the following user query and return only valid pure JSON string:\n{user_query}")
])

chain_classify = prompt_classify | llm

def parse_classification_result(response_text):
        """Clean classification result and parse to JSON dict"""
        cleaned = re.sub(r'^```json\s*|\s*```$', '', response_text.strip(), flags=re.MULTILINE)
        cleaned = re.sub(r'\s+', ' ', cleaned).strip()
        try:
                return json.loads(cleaned)
        except json.JSONDecodeError as e:
                print(f"❌ Classification result parsing failed: {str(e)}, raw response: {response_text[:200]}")
                return {
                        "query_text": response_text,
                        "category": "Other irrelevant questions",
                        "category_code": 0
                }

def classify_user_query(user_query):
        """
        Classify user query
        Parameter: user_query - string, user's query text
        Return: dict, contains query_text, category, category_code
        """
        try:
                response = chain_classify.invoke({"user_query": user_query})
                result = parse_classification_result(response.content)
                result["query_text"] = user_query
                return result
        except Exception as e:
                print(f"❌ Classification API call failed: {str(e)}")
                return {
                        "query_text": user_query,
                        "category": "Other irrelevant questions",
````

```
                    "category": "Other irrelevant questions",
                    "category_code": 0
                }

print("\n=================== Test: User Query Classification ===================")
test_queries = [
        "How much money did I spend in total for these bills?",   # Category 1
        "How much would I have had to pay without the discount?", # Category 2
        "What is the total amount I actually paid?",                    # Category 1
        "What is the total price without discounts?",                  # Category 2
        "How much did I spend on the first receipt?",                  # Category 0 (single receipt)
        "What is the original price of apples?",                          # Category 0 (single item)
        "How is the weather today?",                                           # Category 0 (irrelevant)
        "What is the total discount amount for these receipts?"   # Category 0 (discount amount)
]

for idx, query in enumerate(test_queries, 1):
        classify_result = classify_user_query(query)
        print(f"\n📝 Test Query {idx}: {query}")
        print(f"     📊 Classification Result → Category: {classify_result['category']} | Code: {classify_result['category_code']}")

print("\n=================== Example: Return Answer by Classification ===================")
def get_answer_by_category(classify_result, total_paid, total_original):
        """Return answer based on classification result"""
        code = classify_result["category_code"]
        if code == 1:
                return f"The total paid amount for all receipts is: {round(total_paid, 2)} HKD"
        elif code == 2:
                return f"The total original amount (no discount) for all receipts is: {round(total_original, 2)} HKD"
        else:
                return "Sorry, your question is irrelevant to total paid amount/total original amount (no discount) of receipts, cannot answer."

sample_query = "What is the total amount I actually paid?"
sample_classify = classify_user_query(sample_query)
sample_answer = get_answer_by_category(sample_classify, total_paid_all, total_original_all)
print(f"\n🔍 User Query: {sample_query}")
print(f"📋 Classification Result: {sample_classify['category']}")
print(f"💡 Answer: {sample_answer}")
```

```
=================== Test: User Query Classification ===================

📝 Test Query 1: How much money did I spend in total for these bills?
   📊 Classification Result → Category: Query total paid amount | Code: 1

📝 Test Query 2: How much would I have had to pay without the discount?
   📊 Classification Result → Category: Query total original amount (no discount) | Code: 2

📝 Test Query 3: What is the total amount I actually paid?
   📊 Classification Result → Category: Query total paid amount | Code: 1

📝 Test Query 4: What is the total price without discounts?
   📊 Classification Result → Category: Query total original amount (no discount) | Code: 2

📝 Test Query 5: How much did I spend on the first receipt?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

📝 Test Query 6: What is the original price of apples?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

📝 Test Query 7: How is the weather today?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

📝 Test Query 8: What is the total discount amount for these receipts?
   📊 Classification Result → Category: Other irrelevant questions | Code: 0

=================== Example: Return Answer by Classification ===================

🔍 User Query: What is the total amount I actually paid?
📋 Classification Result: Query total paid amount
💡 Answer: The total paid amount for all receipts is: 1974.3 HKD
```