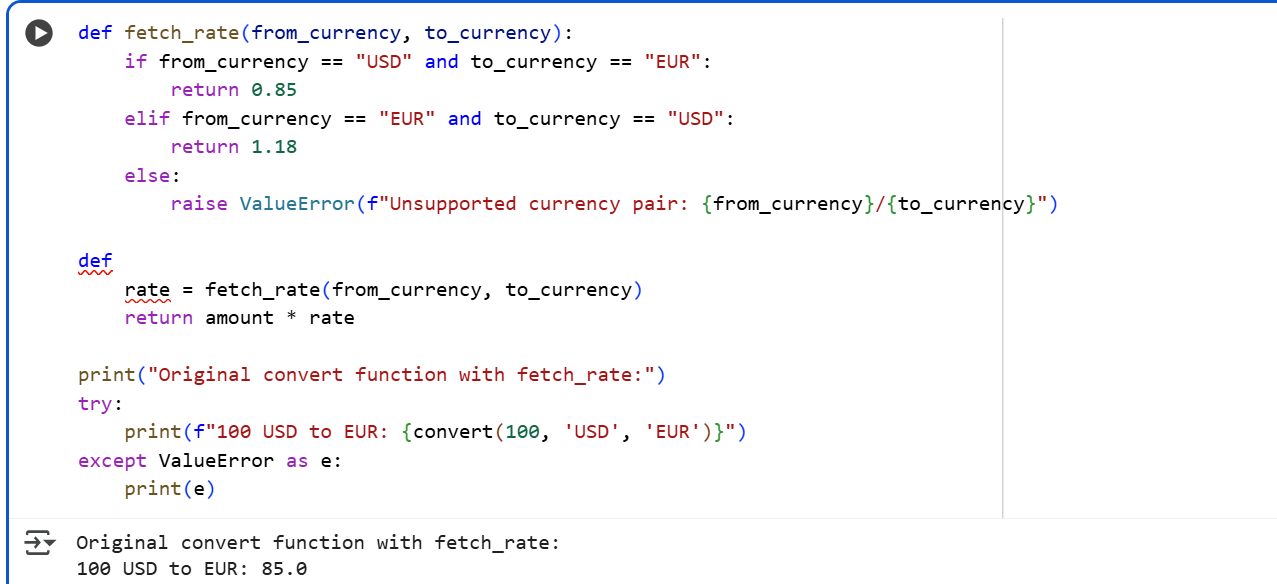
Lab Test-02

Subgroup L  
L.1 — [S05L1] Stub external API for tests  
Scenario (fintech payments):  
Context:  
A currency conversion in fintech payments must be testable without network.  
Your Task:  
Inject a rate-fetch function into `convert(amount, ccy)` and stub it in tests.  
Data & Edge Cases:  
When fetch\_rate('USD')=83.0, convert(10,'USD')=830.0.  
AI Assistance Expectation:  
Use AI to suggest dependency injection or monkeypatch patterns.  
Constraints & Notes:  
Keep convert() pure w.r.t external IO.  
Sample Input  
def convert(amount, ccy): return amount \* fetch\_rate(ccy)  
Sample Output  
convert(10,'USD') with rate 83.0 => 830.0  
Acceptance Criteria: No network; reproducible tests



Explanation:

he flatten\_dict function is a recursive function designed to convert a nested dictionary or list into a single-level dictionary where the keys represent the path to the original values using dot notation for dictionary keys and bracket notation for list indices.

**How it Works:**

The function takes two arguments:

* data: The current nested dictionary or list being processed.
* prefix: A string representing the accumulated key path from the top level down to the current data. This is used to build the flattened key.

**Base Cases:**

The recursion stops when the function encounters a value that is *not* a dictionary or a list. In this case, it means we have reached a leaf node in the nested structure. The function then adds this value to the flat\_data dictionary with the fully constructed key (using the prefix and the current key/index).

**Handling Nested Dictionaries:**

If the data is a dictionary, the function iterates through each key, value pair in the dictionary. For each pair, it constructs a new\_key by appending the current key to the prefix with a dot in between (e.g., prefix.key). If the value is itself a dictionary or a list, the function recursively calls flatten\_dict with the value as the new data and the new\_key as the updated prefix. The results from the recursive call (which is another flattened dictionary for that sub-structure) are then merged into the main flat\_data dictionary using update(). If the value is not a nested structure (i.e., it's a base case), it's directly added to flat\_data with the new\_key.

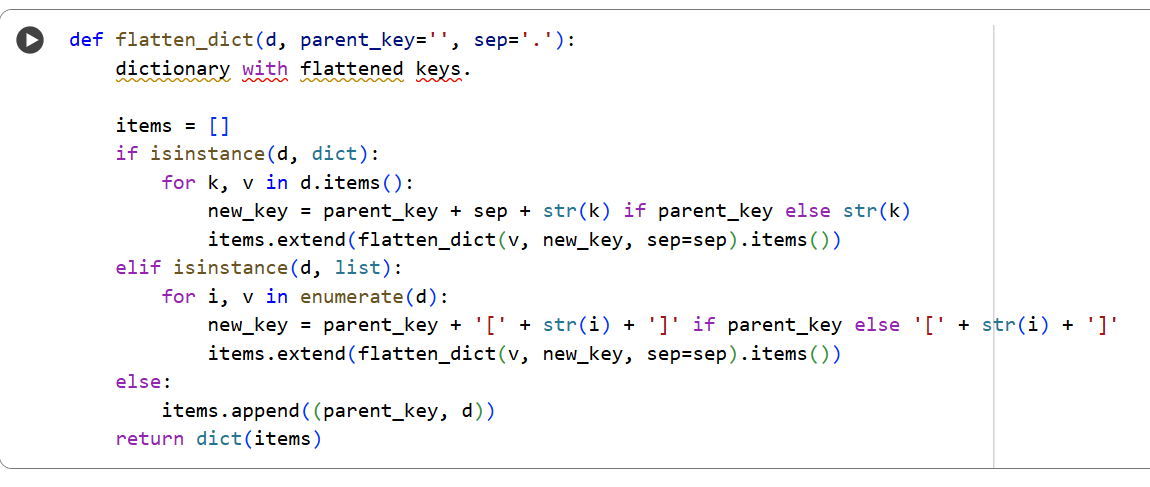
**Handling Nested Lists:**

If the data is a list, the function iterates through each value in the list using its index. For each element, it constructs a new\_key by appending the index in bracket notation to the prefix (e.g., prefix[index]). Similar to dictionaries, if the value is a dictionary or a list, the function recursively calls flatten\_dict with the value as the new data and the new\_key as the updated prefix. The results are merged into flat\_data. If the value is a base case, it's directly added to flat\_data with the new\_key.

**Building the Flattened Keys:**

The prefix argument is crucial for building the flattened keys. It keeps track of the path taken through the nested structure. Each time the function recurses into a nested dictionary or list, the current key or index is appended to the prefix, separated by a dot or enclosed in brackets, respectively. This ensures that the final keys in the flat\_data dictionary accurately represent the original location of the values in the nested structure.

L.2 — [S05L2] Flatten nested JSON with dot keys  
Scenario (fintech payments):  
Context:  
Configs in fintech payments arrive as nested JSON; downstream needs flattened keys.  
Your Task:  
Flatten nested dict to dot-separated keys; for lists, use [index] notation.  
Data & Edge Cases:  
Example provided.  
AI Assistance Expectation:  
Ask AI to propose a recursive function and tests with dict+list combos.  
Constraints & Notes:  
Return a new flat dict.  
Sample Input  
{'user': {'id': 1, 'name': 'Ana'}, 'meta': {'lang': 'en'}}  
Sample Output  
{'user.id':1,'user.name':'Ana','meta.lang':'en'}  
Acceptance Criteria: Handles nested dicts and lists



Explanation:

What the function does

It takes a **nested dictionary** (which may include lists) and returns a **flat dictionary** where:

* Keys are joined using dots (.) for nested dictionaries.
* List items are indexed using square brackets ([0], [1], etc.).
* Input: A **nested dictionary** (configs in fintech).
* Requirement: **Flatten** into dot.separated.keys.
* Return: A new flat dict (do not modify original).