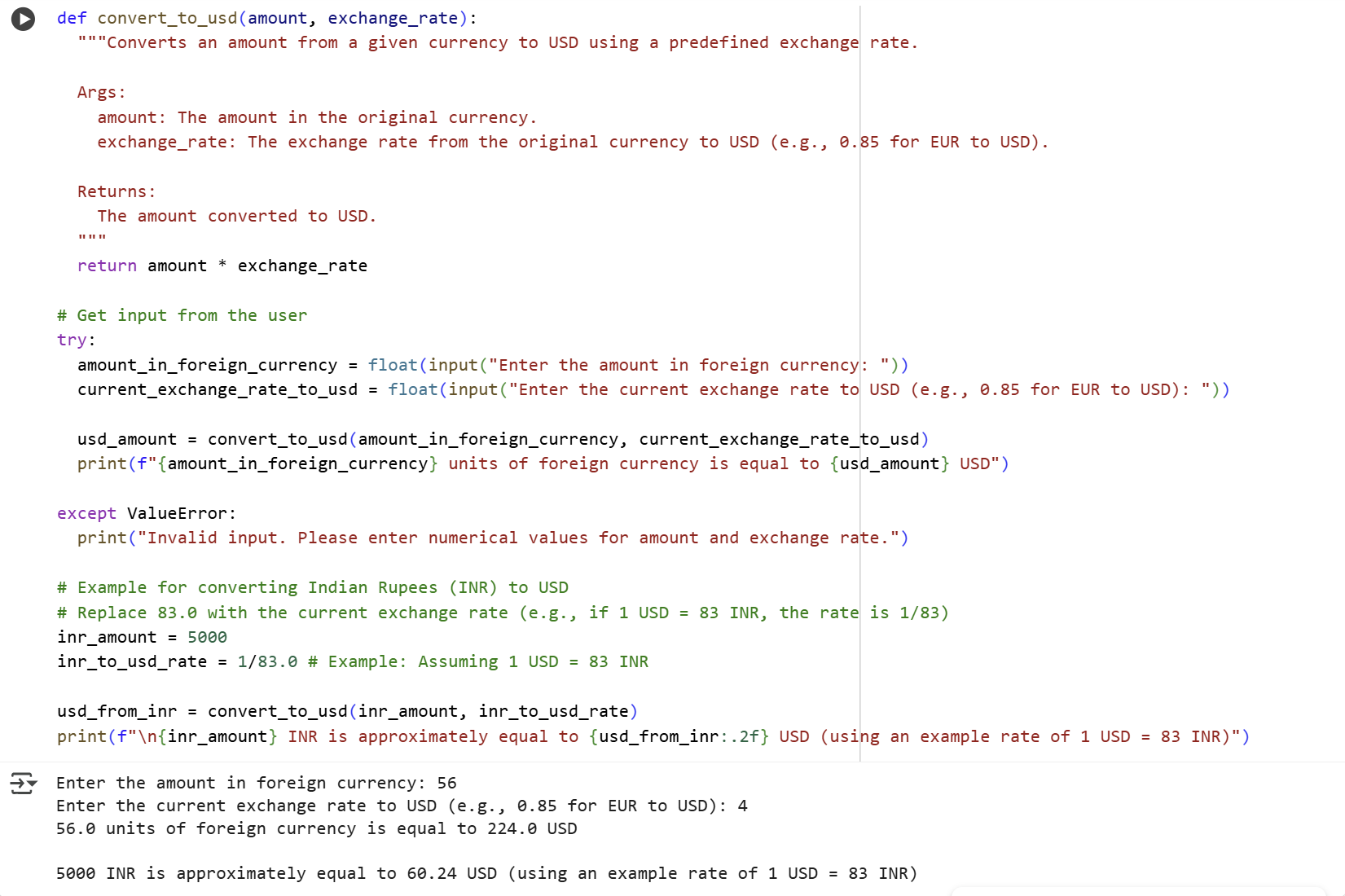
**HTNO:2403A52134**

**Lab Test-2:**

**Task-1: 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:**

def convert\_to\_usd(amount, exchange\_rate):  
  """Converts an amount from a given currency to USD using a predefined exchange rate.  
  
  Args:  
    amount: The amount in the original currency.  
    exchange\_rate: The exchange rate from the original currency to USD (e.g., 0.85 for EUR to USD).  
  
  Returns:  
    The amount converted to USD.  
  """  
  return amount \* exchange\_rate

This block defines a function named convert\_to\_usd that takes two arguments: amount (the value in the foreign currency) and exchange\_rate (the rate to convert the foreign currency to USD). The docstring within triple quotes explains what the function does, its arguments, and what it returns. The line return amount \* exchange\_rate performs the core conversion by multiplying the foreign currency amount by the exchange rate and returning the result.

# Get input from the user  
try:  
  amount\_in\_foreign\_currency = float(input("Enter the amount in foreign currency: "))  
  current\_exchange\_rate\_to\_usd = float(input("Enter the current exchange rate to USD (e.g., 0.85 for EUR to USD): "))  
  
  usd\_amount = convert\_to\_usd(amount\_in\_foreign\_currency, current\_exchange\_rate\_to\_usd)  
  print(f"{amount\_in\_foreign\_currency} units of foreign currency is equal to {usd\_amount} USD")  
  
except ValueError:  
  print("Invalid input. Please enter numerical values for amount and exchange rate.")

This block handles getting input from the user.

* # Get input from the user: This is a comment explaining the purpose of the following code.
* try:: This starts a try block, which is used to handle potential errors (in this case, if the user enters non-numerical input).
* amount\_in\_foreign\_currency = float(input("Enter the amount in foreign currency: ")): This line prompts the user to enter the amount in the foreign currency using the input() function. The entered value is then converted to a floating-point number using float() and stored in the amount\_in\_foreign\_currency variable.
* current\_exchange\_rate\_to\_usd = float(input("Enter the current exchange rate to USD (e.g., 0.85 for EUR to USD): ")): Similar to the previous line, this prompts the user for the exchange rate, converts it to a float, and stores it in current\_exchange\_rate\_to\_usd.
* usd\_amount = convert\_to\_usd(amount\_in\_foreign\_currency, current\_exchange\_rate\_to\_usd): This line calls the convert\_to\_usd function with the user-provided amount and exchange rate, and stores the returned USD amount in the usd\_amount variable.
* print(f"{amount\_in\_foreign\_currency} units of foreign currency is equal to {usd\_amount} USD"): This line prints the result of the conversion using an f-string to format the output nicely.
* except ValueError:: This specifies that if a ValueError occurs within the try block (which happens if float() cannot convert the user's input), the code in the except block will be executed.
* print("Invalid input. Please enter numerical values for amount and exchange rate."): This line prints an error message if the user's input was not valid numerical data.

# Example for converting Indian Rupees (INR) to USD  
# Replace 83.0 with the current exchange rate (e.g., if 1 USD = 83 INR, the rate is 1/83)  
inr\_amount = 5000  
inr\_to\_usd\_rate = 1/83.0 # Example: Assuming 1 USD = 83 INR  
  
usd\_from\_inr = convert\_to\_usd(inr\_amount, inr\_to\_usd\_rate)  
print(f"\n{inr\_amount} INR is approximately equal to {usd\_from\_inr:.2f} USD (using an example rate of 1 USD = 83 INR)")

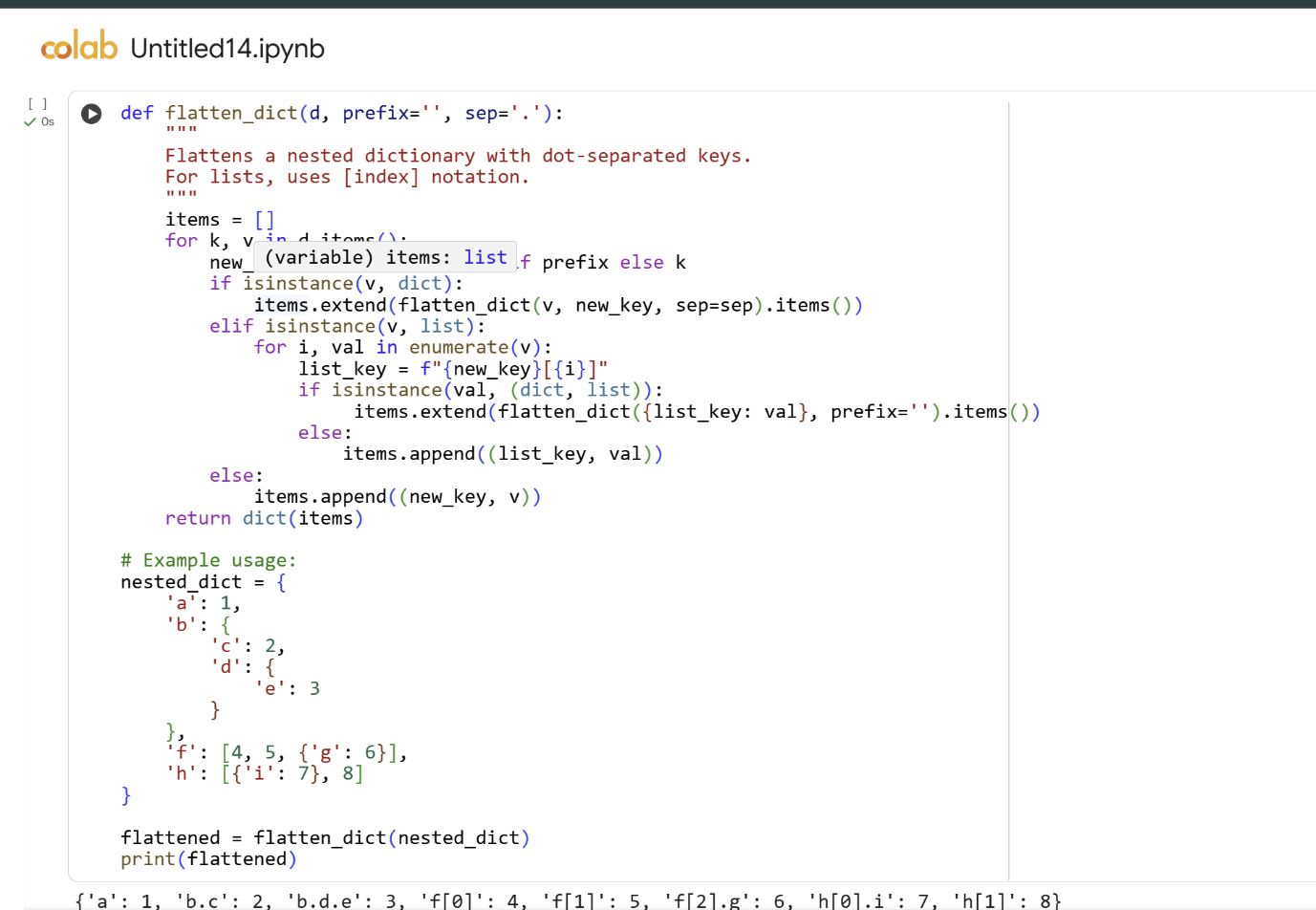
This block provides a specific example of converting Indian Rupees (INR) to USD.

* # Example for converting Indian Rupees (INR) to USD: A comment indicating the purpose of this block.
* # Replace 83.0 with the current exchange rate (e.g., if 1 USD = 83 INR, the rate is 1/83): A comment reminding the user to update the exchange rate.
* inr\_amount = 5000: Sets an example amount in INR.
* inr\_to\_usd\_rate = 1/83.0 # Example: Assuming 1 USD = 83 INR: Sets an example exchange rate for INR to USD. The comment explains the basis of this rate.
* usd\_from\_inr = convert\_to\_usd(inr\_amount, inr\_to\_usd\_rate): Calls the convert\_to\_usd function with the example INR amount and exchange rate.
* print(f"\n{inr\_amount} INR is approximately equal to {usd\_from\_inr:.2f} USD (using an example rate of 1 USD = 83 INR)"): Prints the result of the INR conversion example, formatted to two decimal places using :.2f. The \n at the beginning adds a new line for better readability.

**Task-2:**

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

Code&Output:



**Explanation**:

def flatten\_dict(d, prefix='', sep='.'):

This line defines a function named flatten\_dict. It takes three arguments:

* d: The dictionary to flatten.
* prefix: A string that will be used as a prefix for the keys in the flattened dictionary. It defaults to an empty string.
* sep: The separator character to use between nested keys. It defaults to a dot (.).

    """  
    Flattens a nested dictionary with dot-separated keys.  
    For lists, uses [index] notation.  
    """

This is a docstring, which explains what the function does. It mentions flattening a nested dictionary using dot-separated keys and using [index] notation for lists.

    items = []

This line initializes an empty list called items. This list will temporarily store the key-value pairs of the flattened dictionary as tuples (key, value).

    for k, v in d.items():

This line starts a for loop that iterates through each key-value pair (k, v) in the input dictionary d.

        new\_key = prefix + sep + k if prefix else k

This line constructs the new\_key for the flattened dictionary.

* If prefix is not empty (meaning we are inside a nested structure), it concatenates the prefix, the sep character, and the current key k.
* If prefix is empty (meaning we are at the top level), the new\_key is just the current key k.

        if isinstance(v, dict):

This line checks if the current value v is an instance of a dictionary.

            items.extend(flatten\_dict(v, new\_key, sep=sep).items())

If v is a dictionary, this line recursively calls the flatten\_dict function with the nested dictionary v. The new\_key calculated earlier becomes the prefix for the recursive call. The sep is passed along. The .items() method is called on the result of the recursive call to get its key-value pairs as tuples, and extend() adds these tuples to the items list.

        elif isinstance(v, list):

This line checks if the current value v is an instance of a list.

            for i, val in enumerate(v):

If v is a list, this line starts a nested for loop that iterates through each element val in the list v, along with its index i using enumerate().

                list\_key = f"{new\_key}[{i}]"

Inside the list loop, this line constructs the key for the list item. It takes the new\_key (which represents the path to the list) and appends [i] (the index of the current item in the list) using an f-string.

                if isinstance(val, (dict, list)):

This line checks if the current list item val is either a dictionary or a list itself (i.e., a nested structure within the list).

                     items.extend(flatten\_dict({list\_key: val}, prefix='').items())

If val is a nested dictionary or list, this line handles it. It creates a temporary dictionary {list\_key: val} where the key is the constructed list\_key and the value is the nested structure. It then recursively calls flatten\_dict on this temporary dictionary. The prefix for this recursive call is set to an empty string because the list\_key already contains the full path including the list notation. The .items() are extended to the items list.

                else:  
                    items.append((list\_key, val))

If the current list item val is not a dictionary or list (i.e., a simple value), this line appends a tuple containing the list\_key and the value val to the items list.

        else:

This line is the else part of the initial if-elif chain, meaning the current value v is neither a dictionary nor a list.

            items.append((new\_key, v))

If v is a simple value, this line appends a tuple containing the constructed new\_key and the value v to the items list.

    return dict(items)

After all key-value pairs in the input dictionary d (and any nested structures) have been processed, this line converts the items list (which contains key-value tuples) back into a dictionary and returns it.

# Example usage:  
nested\_dict = {  
    'a': 1,  
    'b': {  
        'c': 2,  
        'd': {  
            'e': 3  
        }  
    },  
    'f': [4, 5, {'g': 6}],  
    'h': [{'i': 7}, 8]  
}

This block is an example of how to use the flatten\_dict function. It defines a sample nested dictionary called nested\_dict.

flattened = flatten\_dict(nested\_dict)

This line calls the flatten\_dict function with the nested\_dict and stores the returned flattened dictionary in the flattened variable.

print(flattened)

This line prints the content of the flattened dictionary.