Exercise One Answer AND Exercise Two Question One Answer

Exercise 1: Creating Property Listings with Python Dictionaries (at home?)

Goal: Practice core data structures and build foundational knowledge for the recommender system.

Concepts Covered:

- Variables, types, and basic I/O
- Lists and dictionaries
- Looping and conditional logic

Task:

1. Define 5 sample property listings as Python dictionaries. Each dictionary should include:

o property_id (unique string or integer)

o location (string)

o type (e.g., "cabin", "condo")

o price_per_night (numeric)

o features (list of strings)

o tags (list of strings) 2. Store these dictionaries in a list called property_listings. 3. Write a loop that:

o Prints each listing in a readable format.

o Filters and prints properties under a given budget.

```
propertyone = dict()
propertyone['property_id']= 1
propertyone['location'] = "Alberta"
propertyone['type'] = "cabin"
propertyone['price_per_night'] = 299
propertyone['features'] = ["Size", "Level", "Floor Space"]
propertyone['tags'] = ["automatic", "efficient"]
print(propertyone)
```

```
{'property_id': 1, 'location': 'Alberta', 'type': 'cabin', 'price_per_night': 299, 'features': ['Size', 'Level', 'Floor Space'], 'tags': ['automatic', 'efficient']}
```

```
propertytwo = dict()
propertytwo['property_id']= 2
propertytwo['location'] = "Toronto"
propertytwo['type'] = "condo"
propertytwo['price_per_night'] = 399
propertytwo['features'] = ["Three Bedrooms", "Two Kitchens"]
propertytwo['tags'] = ["Great Sunshine", "Wonderful View"]
print(propertytwo)
```

```
{'property_id': 2, 'location': 'Toronto', 'type': 'condo', 'price_per_night': 399, 'features': ['Three Bedrooms', 'Two Kitchens'], 'tags': ['Great Sunshine', 'Wonderful View']}
```

```
propertythree = dict()
propertythree['property_id']= 3
propertythree['location'] = "British Columbia"
propertythree['type'] = "condo"
propertythree['price_per_night'] = 499
propertythree['features'] = ["Two Bedrooms", "Two Kitchens"]
propertythree['tags'] = ["Luxury Condo", "Modern Design"]
print(propertythree)
```

```
{'property_id': 3, 'location': 'British Columbia', 'type': 'condo', 'price_per_night': 499, 'features': ['Two Bedrooms', 'Two Kitchens'], 'tags': ['Luxury Condo', 'Modern Design']}
```

```
propertyfour = dict()
propertyfour['property_id']= 4
propertyfour['location'] = "Nova Scotia,"
propertyfour['type'] = "house"
propertyfour['price_per_night'] = 199
propertyfour['features'] = ["One Bedrooms", "Shared Kitchens"]
propertyfour['tags'] = ["Pet Friendly", "Free Parking"]
print(propertyfour)
```

```
{'property_id': 4, 'location': 'Nova Scotia,', 'type': 'house', 'price_per_night': 199, 'features': ['One Bedrooms', 'Shared Kitchens'], 'tags': ['Pet Friendly', 'Free Parking']}
```

```
propertyfive = dict()
propertyfive['property_id']= 5
propertyfive['location'] = "Quebec"
propertyfive['type'] = "condo"
propertyfive['price_per_night'] = 499
propertyfive['features'] = ["Four Bedrooms", "Four Kitchens"]
propertyfive['tags'] = ["Closed to trnasit", "Luxury Design"]
print(propertyfive)
```

```
{'property_id': 5, 'location': 'Quebec', 'type': 'condo', 'price_per_night': 499, 'features': ['Four Bedrooms', 'Four Kitchens'], 'tags': ['Closed to trnasit', 'Luxury Design']}
```

```
property_listings = [propertyone, propertytwo, propertythree, propertyfour, propertyfive]
```

```
#Question Three: Using Looping: Display property listing in a readable format, either the gross format, or a table
print(property_listings)
```

```
[{'property_id': 1, 'location': 'Alberta', 'type': 'cabin', 'price_per_night': 299, 'features': ['Size', 'Level', 'Floor Space'], 'tags': ['automatic', 'efficient']}, {'property_id': 2, 'location': 'To
```

```
for proper in property_listings:
    print(f"Property ID:  {proper['property_id']}")
    print(f"Location:  {proper['location']}")
    print(f"Type:  {proper['type']}")
    print(f"Price:  {proper['price_per_night']}")
    print(f"Feature:  {proper['features']}")
    print(f"tags:  {proper['tags']}")
    print("-" * 40)
```

```
Property ID:  1
Location:  Alberta
Type:  cabin
Price:  299
Feature:  ['Size', 'Level', 'Floor Space']
tags:  ['automatic', 'efficient']
----------------------------------------
Property ID:  2
Location:  Toronto
Type:  condo
Price:  399
Feature:  ['Three Bedrooms', 'Two Kitchens']
tags:  ['Great Sunshine', 'Wonderful View']
----------------------------------------
Property ID:  3
Location:  British Columbia
Type:  condo
Price:  499
Feature:  ['Two Bedrooms', 'Two Kitchens']
tags:  ['Luxury Condo', 'Modern Design']
----------------------------------------
Property ID:  4
Location:  Nova Scotia,
Type:  house
Price:  199
Feature:  ['One Bedrooms', 'Shared Kitchens']
tags:  ['Pet Friendly', 'Free Parking']
----------------------------------------
Property ID:  5
Location:  Quebec
Type:  condo
Price:  499
Feature:  ['Four Bedrooms', 'Four Kitchens']
tags:  ['Closed to trnasit', 'Luxury Design']
----------------------------------------
```

```
from tabulate import tabulate
print(tabulate(property_listings, headers='keys', tablefmt='psql'))
```

```
+---------------+-----------------+--------+-------------------+-----------------------------------+-----------------------------------------+
|   property_id | location        | type   |   price_per_night | features                          | tags                                    |
|---------------+-----------------+--------+-------------------+-----------------------------------+-----------------------------------------|
|             1 | Alberta         | cabin  |               299 | ['Size', 'Level', 'Floor Space']  | ['automatic', 'efficient']              |
|             2 | Toronto         | condo  |               399 | ['Three Bedrooms', 'Two Kitchens'] | ['Great Sunshine', 'Wonderful View']   |
|             3 | British Columbia | condo  |               499 | ['Two Bedrooms', 'Two Kitchens']  | ['Luxury Condo', 'Modern Design']       |
|             4 | Nova Scotia,    | house  |               199 | ['One Bedrooms', 'Shared Kitchens'] | ['Pet Friendly', 'Free Parking']      |
|             5 | Quebec          | condo  |               499 | ['Four Bedrooms', 'Four Kitchens'] | ['Closed to trnasit', 'Luxury Design'] |
+---------------+-----------------+--------+-------------------+-----------------------------------+-----------------------------------------+
```

双击（或按回车键）即可修改

```
import pandas as pd
```

```
#Question Three: Display property listing in a readable format, solution method is using a table in Pandas
print(property_listings)
dataframe = pd.DataFrame(property_listings)
print(dataframe)
```

```
[{'property_id': 1, 'location': 'Alberta', 'type': 'cabin', 'price_per_night': 299, 'features': ['Size', 'Level', 'Floor Space'], 'tags': ['automatic', 'efficient']}, {'property_id': 2, 'location': 'To
   property_id          location   type  price_per_night  \
0            1           Alberta  cabin              299
1            2           Toronto  condo              399
2            3  British Columbia  condo              499
3            4      Nova Scotia,  house              199
4            5            Quebec  condo              499

                       features                        tags
0       [Size, Level, Floor Space]              [automatic, efficient]
1   [Three Bedrooms, Two Kitchens]    [Great Sunshine, Wonderful View]
2    [Two Bedrooms, Two Kitchens]       [Luxury Condo, Modern Design]
3  [One Bedrooms, Shared Kitchens]         [Pet Friendly, Free Parking]
4   [Four Bedrooms, Four Kitchens]  [Closed to trnasit, Luxury Design]
```

```
#Question Three part two: Filters and prints properties under a given budgt
```

```
filtered_list = []

for property_item in property_listings:
    if property_item['price_per_night'] <= 400:  #Filter the price of condo by the budget under $400.
        filtered_list.append(property_item)
print(f"Filtered (for loop): {filtered_list}")
```

```
Filtered (for loop): [{'property_id': 1, 'location': 'Alberta', 'type': 'cabin', 'price_per_night': 299, 'features': ['Size', 'Level', 'Floor Space'], 'tags': ['automatic', 'efficient']}, {'property_i
```

filter property under a given budget:

```
budget = 400
print(f"Properties is under ${budget} per night：")
print("=" * 40)
for proper in property_listings:
    if proper['price_per_night'] <= budget:
        print(f"{proper['location']} - ${proper['price_per_night']}")
```

```
Properties is under $400 per night：
========================================
Alberta - $299
Toronto - $399
Nova Scotia, - $199
```

Finally, this is an example how users input a tags and find the matching property in engine

```
search_tags = input("Enter a tags to search for: ").strip().lower()
print(f"Properties tagsged with '{search_tags}':")
print("=" * 40)
for proper in property_listings:
    if search_tags in [tags.lower() for tags in proper['tags']]:
        print(f"{proper['location']} - {proper['price_per_night']} - tags: {'， '.join(proper['tags'])}")
        #print(f"{proper['location']} - {proper['price_per_night']} - tags: {'， '.join(proper['tags'])}")
```

```
Enter a tags to search for: pet friendly
Properties tagsged with 'pet friendly':
========================================
Nova Scotia, - 199 - tags: Pet Friendly， Free Parking
```

Goal: Apply OOP principles to the project and introduce persistent storage.

Concepts Covered:

- Classes and methods
- JSON file handling
- SQLite with sqlite3

Task Part A: Classes

1. Define a Property class with attributes:

o property_id, location, type, price_per_night, features, tags 2. Define a User class with:

o user_id, name, group_size, preferred_environment, budget

o Method: matches(self, property_obj) – returns True if property is a good fit

Code Written In Class With help of Colab Copliot

```
class Property:
  def __init__(self, property_id, location, type, price_per_night, features, tags):
    self.property_id = property_id
    self.location = location
    self.type = type
    self.price_per_night = price_per_night
    self.features = features
    self.tags = tags
```

```
class User:
  def __init__(self, user_id, name, group_size, preferred_enviornment, budget):
    self.user_id = user_id
    self.name = name
    self.group_size = group_size
    self.preferred_enviornment = preferred_enviornment
```

Goal: Apply OOP principles to the project and introduce persistent storage.

Task Part A: Classes

1. Define a Property class with attributes:

o property_id, location, type, price_per_night, features, tags 2. Define a User class with:

o user_id, name, group_size, preferred_environment, budget

o Method: matches(self, property_obj) – returns True if property is a good fit

Excercise Two Part Two

Task Part B: JSON File I/O

1. Create several Property and User objects and convert them to dictionaries.
2. Save these to JSON files using json.dump().
3. Load and recreate the objects from the files using json.load().

Refined Version on Exercise Two Part A

```
import json
import sqlite3
import os
from sqlite_example import print_table
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipython-input-2645411178.py in <cell line: 0>()
      2 import sqlite3
      3 import os
----> 4 from sqlite_example import print_table

ModuleNotFoundError: No module named 'sqlite_example'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
### Part A: Class Definitions ===
class Property:
    def __init__(self, property_id, location, type_, price_per_night, features, tags):
        self.property_id = property_id
        self.location = location
        self.type = type_
        self.price_per_night = price_per_night
        self.features = features
        self.tags = tags

    def to_dict(self):
        return {
            "property_id": self.property_id,
            "location": self.location,
            "type": self.type,  # keep JSON field as "type"
            "price_per_night": self.price_per_night,
            "features": self.features,
            "tags": self.tags,
        }
# Function Defined: Create several python instant objects and store them into the dictionary
    @classmethod
    def from_dict(cls, d):
# Function Defined: from_dic() function takes dictionary as input and return the property class instance as output

        return cls(
            property_id=d["property_id"],
            location=d["location"],
            type_=d["type"],
            price_per_night=d["price_per_night"],
            features=d["features"],
            tags=d["tags"],
        )
# Recreate the new instance from property_class from the dictionary, from one way to another way, verse versa
```

```
class User:
    def __init__(self, user_id, name, group_size, preferred_environment, budget):
        self.user_id = user_id
        self.name = name
        self.group_size = group_size
        self.preferred_environment = preferred_environment
        self.budget = budget

    def matches(self, property_obj):
        """Returns True if property is within budget and preferred environment."""
        return (
            property_obj.price_per_night <= self.budget and
            self.preferred_environment in property_obj.tags
        )

    def to_dict(self):
        return {
            "user_id": self.user_id,
            "name": self.name,
            "group_size": self.group_size,
            "preferred_environment": self.preferred_environment,
            "budget": self.budget,
        }

    @classmethod
    def from_dict(cls, d):
        return cls(
            user_id=d["user_id"],
            name=d["name"],
            group_size=d["group_size"],
            preferred_environment=d["preferred_environment"],
            budget=d["budget"]
        )
```

```
# === Sample Data ===

properties = [
    Property(1, "Alberta", "cabin", 180, ["Size", "Level", "Floor Space"], ["automatic", "efficient"]),
    Property(2, "Toronto", "condo", 399, ["Three Bedrooms", "Two Kitchens"], ["Great Sunshine", "Wonderful View"]),
    Property(3, "British Columbia", "condo", 499, ["Two Bedrooms", "Two Kitchens"], ["Luxury Condo", "Modern Design"]),
    Property(4, "Nova Scotia,", "house", 199, ["One Bedrooms", "Shared Kitchens"], ["Pet Friendly", "Free Parking"]),
]
users = [
    User(101, "Alice", 4, "lake", 200),
    User(102, "Bob", 2, "mountain", 150),
    User(103, "Charlie", 5, "city", 300)
]
```

Excercise Two Part Two

Task Part B: JSON File I/O

1. Create several Property and User objects and convert them to dictionaries.
2. Save these to JSON files using json.dump().
3. Load and recreate the objects from the files using json.load().

```
# Create five propoerty objects
propertyone = Property(1, "Alberta", "cabin", 299, ["Size", "Level", "Floor Space"], ["automatic", "efficient"])
propertytwo = Property(2, "Toronto", "condo", 399, ["Three Bedrooms", "Two Kitchens"], ["Great Sunshine", "Wonderful View"])
propertythree = Property(3, "British Columbia", "condo", 499, ["Two Bedrooms", "Two Kitchens"], ["Luxury Condo", "Modern Design"])
propertyfour = Property(4, "Nova Scotia,", "house", 199, ["One Bedrooms", "Shared Kitchens"], ["Pet Friendly", "Free Parking"])
propertyfive = Property(5, "Quebec", "condo", 499, ["Four Bedrooms", "Four Kitchens"], ["Closed to trnasit", "Luxury Design"])
# Store each objects into dictionaries
propertyone_dict = propertyone.__dict__
propertytwo_dict = propertytwo.__dict__
propertythree_dict = propertythree.__dict__
propertyfour_dict = propertyfour.__dict__
propertyfive_dict = propertyfive.__dict__
# Print five dictionaries
print(propertyone_dict)
print(propertytwo_dict)
print(propertythree_dict)
print(propertyfour_dict)
print(propertyfive_dict)
```

```
{'property_id': 1, 'location': 'Alberta', 'type': 'cabin', 'price_per_night': 299, 'features': ['Size', 'Level', 'Floor Space'], 'tags': ['automatic', 'efficient']}
{'property_id': 2, 'location': 'Toronto', 'type': 'condo', 'price_per_night': 399, 'features': ['Three Bedrooms', 'Two Kitchens'], 'tags': ['Great Sunshine', 'Wonderful View']}
{'property_id': 3, 'location': 'British Columbia', 'type': 'condo', 'price_per_night': 499, 'features': ['Two Bedrooms', 'Two Kitchens'], 'tags': ['Luxury Condo', 'Modern Design']}
{'property_id': 4, 'location': 'Nova Scotia,', 'type': 'house', 'price_per_night': 199, 'features': ['One Bedrooms', 'Shared Kitchens'], 'tags': ['Pet Friendly', 'Free Parking']}
{'property_id': 5, 'location': 'Quebec', 'type': 'condo', 'price_per_night': 499, 'features': ['Four Bedrooms', 'Four Kitchens'], 'tags': ['Closed to trnasit', 'Luxury Design']}
```

```
# Create five users objects
userone = User(101, "Alice", 4, "lake", 200)
usertwo = User(102, "Bob", 2, "mountain", 150)
userthree = User(103, "Charlie", 5, "city", 300)
userfour = User(104, "Diana", 3, "beach", 250)
userfive = User(105, "Ethan", 6, "forest", 180)
# Store each objects into dictionaries
```

```
userone_dict = userone.__dict__
usertwo_dict = usertwo.__dict__
userthree_dict = userthree.__dict__
userfour_dict = userfour.__dict__
userfive_dict = userfive.__dict__
# Print five dictionaries
print(userone_dict)
print(usertwo_dict)
print(userthree_dict)
print(userfour_dict)
print(userfive_dict)
```

```
{'user_id': 101, 'name': 'Alice', 'group_size': 4, 'preferred_enviornment': 'lake'}
{'user_id': 102, 'name': 'Bob', 'group_size': 2, 'preferred_enviornment': 'mountain'}
{'user_id': 103, 'name': 'Charlie', 'group_size': 5, 'preferred_enviornment': 'city'}
{'user_id': 104, 'name': 'Diana', 'group_size': 3, 'preferred_enviornment': 'beach'}
{'user_id': 105, 'name': 'Ethan', 'group_size': 6, 'preferred_enviornment': 'forest'}
```

```
#Converting Python Objects to JSON Strings. We can use `json.dumps()` to convert a Python object into a JSON-formatted string.
import json

# Example lists of dictionaries
property_dicts = [
    propertyone_dict,
    propertytwo_dict,
    propertythree_dict,
    propertyfour_dict,
    propertyfive_dict
]

user_dicts = [
    userone_dict,
    usertwo_dict,
    userthree_dict,
    userfour_dict
]

# Convert and print property JSON strings
for i, prop_dict in enumerate(property_dicts):
    prop_json_string = json.dumps(prop_dict, indent=4)
    print(prop_json_string)

print()

# Convert and print user JSON strings
for i, user_dict in enumerate(user_dicts):
    user_json_string = json.dumps(user_dict, indent=4)
    print(user_json_string)
```

```
    "tags": [
        "Luxury Condo",
        "Modern Design"
    ]
}
{
    "property_id": 4,
    "location": "Nova Scotia,",
```

```
{
    "user_id": 103,
    "name": "Charlie",
    "group_size": 5,
    "preferred_enviornment": "city"
}
{
    "user_id": 104,
    "name": "Diana",
    "group_size": 3,
    "preferred_enviornment": "beach"
}
```

```
# Load property from file
for prop_dict in property_dicts:
    with open("property.json", "w") as f:
        json.dump(prop_dict, f)
with open("property.json", "r") as f:
    loaded_property = json.load(f)

for user_dict in user_dicts:
    with open("user.json", "w") as f:
        json.dump(user_dict, f)
# Load user from file
with open("user.json", "r") as f:
    loaded_user = json.load(f)

print(loaded_property)
print(loaded_user)
```

```
{'property_id': 5, 'location': 'Quebec', 'type': 'condo', 'price_per_night': 499, 'features': ['Four Bedrooms', 'Four Kitchens'], 'tags': ['Closed to trnasit', 'Luxury Design']}
{'user_id': 104, 'name': 'Diana', 'group_size': 3, 'preferred_enviornment': 'beach'}
```

Refined Version on Exercise Two Part B

```
# Save Data to JSON
with open("properties.json", "w") as f:
    json.dump([p.to_dict() for p in properties], f, indent=2)
with open("users.json", "w") as f:
    json.dump([u.to_dict() for u in users], f, indent=2)
# Using the newly defined function to save python dictionary file into json file

# Load and reconstruct using from_dict
with open("properties.json", "r") as f:
    loaded_properties = [Property.from_dict(d) for d in json.load(f)]

with open("users.json", "r") as f:
    loaded_users = [User.from_dict(d) for d in json.load(f)]
# Using the newly defined function from.dict() to Unpack JSON file into Python dictionary, then python object instance

# Test matching logic-match the tags in properties with the users'environment preference


print("\nMatching Results:")
for user in loaded_users:
    for prop in loaded_properties:
        if user.matches(prop):
            print(f"{user.name} matches with {prop.type} in {prop.location} (${prop.price_per_night}/night)")

# 💯 💯 💯 💯 💯
```

```
Matching Results:
```

Task Part C: SQLite

1. Create a SQLite database rentals.db with two tables: users and properties.
2. Insert the sample data from Part A.
3. Write a query to retrieve all properties under a user's budget.

```
import sqlite3


# Connect (or create) the SQLite database
conn = sqlite3.connect("rentals.db")
cursor = conn.cursor()

# Create users table
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    budget INTEGER NOT NULL
)
''')

# Create properties table
cursor.execute('''
```

```
CREATE TABLE IF NOT EXISTS properties (
    property_id INTEGER PRIMARY KEY,
    location TEXT NOT NULL,
    type TEXT NOT NULL,
    price_per_night INTEGER NOT NULL
)
''')

# Insert data into users
cursor.executemany('INSERT OR IGNORE INTO users (user_id, name, budget) VALUES (?, ?, ?)', users_data)

# Insert data into properties
cursor.executemany('INSERT OR IGNORE INTO properties (property_id, location, type, price_per_night) VALUES (?, ?, ?, ?)', properties_data)

conn.commit()
```

```
user_name = "Alice"

cursor.execute('''
SELECT p.property_id, p.location, p.type, p.price_per_night
FROM properties p
JOIN users u ON u.user_id = ?
WHERE p.price_per_night <= u.budget
''', (1,))  # user_id for Alice is 1

results = cursor.fetchall()

print(f"Properties under {user_name}'s budget:")
for row in results:
    print(row)

conn.close()
```

Course project are dealing with small sample sized database users, still conformtable for using SQLite

SQLite is a lightweight, file-based database engine that comes built into Python.

We use the built-in 'sqlite3' module to work with it, and the 'tabulate' package

to print query results in a readable table

This example:

1. connects to (or Creats) a database file
2. Creates a 'properties' table
3. Inserts data
4. Runs a few updates/deletes
5. Prints the table after each modification using tabulate

```
import sqlite3
from tabulate import tabulate

def print_table(cursor, title=None, table_name="properties"):
  """
  #pretty-print the content of a tbale using tabulate.
  #NOTE: table_name is used directly in SQL here becuase it's a trusted identifier.
  #Do Not pass untrusted user input as table_name.
  """
  if title:
    print(f"\n{title}")
  cursor.execute(f"SELECT * FROM {table_name}")
  rows = cursor.fetchall()
  headers = [desc[0] for desc in cursor.description]
  if rows:
    print(tabulate(rows, headers=headers, tablefmt="grid"))
  else:
    print("(table is empty)")
```

```
if __name__ == "__main__":
    # 1. Connect to (or create) a database file
    #    If 'example.db' does not exist, SQLite will create it.
    #    Think of this as "opening" or "making" a spreadsheet file to store data.
    conn = sqlite3.connect("example.db")

    # 2. Create a cursor object — used to send SQL commands to the database
    cursor = conn.cursor()
```

```
cursor = conn.cursor()
```

```
# 3. Create a table (if it doesn't already exist)
  #    In a database, a TABLE is like a spreadsheet.
  #    Each COLUMN defines a type of information we store (like "location" or "price_per_night"),
  #    and each ROW will be one record (one property) in the table.
  #    Here, we create a table named 'properties' with the following columns:
  #      - property_id: INTEGER, serves as the PRIMARY KEY (unique ID for each property)
  #      - location: TEXT (string) — where the property is located
  #      - type: TEXT (string) — type of property (e.g., "cabin", "apartment")
  #      - price_per_night: REAL (floating-point number) — nightly rental price
  #
  #    Example layout after inserting some data:
  #    -----------------------------------------------------------
  #    | property_id | location |   type   | price_per_night   |
  #    |-------------|----------|----------|-------------------|
  #    |      1      |  Banff   |  cabin   |       180.0       |
  #    |      2      | Toronto  | apartment|       120.0       |
  #    -----------------------------------------------------------
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS properties (
    property_id INTEGER PRIMARY KEY,
    location TEXT NOT NULL,
    type TEXT NOT NULL,
    price_per_night REAL NOT NULL
)
""")
conn.commit()
print_table(cursor, "After (potential) table creation:")

#4. (Optional) Clear existing rows so our demo always starts fresh

cursor.execute("DELETE FROM properties")
conn.commit()
print_table(cursor, "After clearing exitsing rows(demo reset):")
```

```
After (potential) table creation:
+---------------+------------+--------+-------------------+
|   property_id | location   | type   |   price_per_night |
+===============+============+========+===================+
|             1 | Alberta    | cabin  |               180 |
+---------------+------------+--------+-------------------+
|             2 | Toronto    | condo  |               377 |
+---------------+------------+--------+-------------------+

After clearing exitsing rows(demo reset):
(table is empty)
```

```
#5. Insert some rows into the atbel
# Each INSERT adds one ROW (record) to the table.
# The VALUES (?, ?, ?) part is a placeholer pattern - the quetsion marks
# get replaced by actual values provided in the tuple argument.
```

```
cursor.execut("""
INSERRT INTO properties (location, type, price_per_night)
VALUES (?, ?, ?)
""", ("Alberta", "cabin", 180))
conn.commit()
print_table(cursor, "After inserting Alberta:")

cursor.execut("""
INSERRT INTO properties (location, type, price_per_night)
VALUES (?, ?, ?)
""", ("Toronto", "condo", 399))
conn.commit()
print_table(cursor, "After inserting Toronto:")
```

```
After inserting Alberta:
+---------------+------------+--------+-------------------+
|   property_id | location   | type   |   price_per_night |
+===============+============+========+===================+
|             1 | Alberta    | cabin  |               180 |
+---------------+------------+--------+-------------------+

After inserting Toronto:
+---------------+------------+--------+-------------------+
|   property_id | location   | type   |   price_per_night |
+===============+============+========+===================+
|             1 | Alberta    | cabin  |               180 |
+---------------+------------+--------+-------------------+
|             2 | Toronto    | condo  |               399 |
+---------------+------------+--------+-------------------+
```

```
#6. Query (read from) the table
# SELECT * means "get all columns for all rows".
# The result will be a list of tuples, where each tuple is one row form the table

cursor.execute("SELECT * FROM properties")
rows = cursor.fetchall()
print("\nQuery result {(raw tuples):")
print(rows)
# For a nicer view, we cna also use our helper function:
print_table(cursor, "Current full table after initial function:")



#7. Example UPDATE: modify existing data
# Here we change Toronto's nightly price from 399.0 to 377.0
cursor.execute("""
UPDATE properties
SET price_per_night = ?
WHERE location = ?
""", (377.0, "Toronto"))
conn.commit()
print_table(cursor, "After updating Toronto's price to 377.0:")
```

```
Query result {(raw tuples):
[(1, 'Alberta', 'cabin', 180.0), (2, 'Toronto', 'condo', 399.0)]

Current full table after initial function:
+---------------+------------+--------+-------------------+
|   property_id | location   | type   |   price_per_night |
+===============+============+========+===================+
|             1 | Alberta    | cabin  |               180 |
+---------------+------------+--------+-------------------+
|             2 | Toronto    | condo  |               399 |
+---------------+------------+--------+-------------------+

After updating Toronto's price to 377.0:
+---------------+------------+--------+-------------------+
|   property_id | location   | type   |   price_per_night |
+===============+============+========+===================+
|             1 | Alberta    | cabin  |               180 |
+---------------+------------+--------+-------------------+
|             2 | Toronto    | condo  |               377 |
+---------------+------------+--------+-------------------+
```

Task Part C: SQLite

1. Create a SQLite database rentals.db with two tables: users and properties.
2. Insert the sample data from Part A.
3. Write a query to retrieve all properties under a user’s budget.

Refined Version on Exercise Two Part C

```
# === Part C: SQLite Integration ===

db_file = "rentals.db"

# If the database file already exists, delect it (this removes all tables $ data)
if os.path.exists(db_file):
    os.remove(db_file)
    print("Database resetL old file deleted.")
# Create SQLite DB

conn = sqlite3.connect(db_file)
cur = conn.cursor()

#Create atbles
cur.execute("""
CREATE TABLE IF NOT EXISTS users (
  user_id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  group_size INTEGER NOT NULL,
  preferred_environment TEXT NOT NULL,
  budget REAL NOT NULL
)
""")
# Commit any pending changes so schema is updated

conn.commit()

# Check all tables in the database

cur.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cur.fetchall()
print("Tables in the database:", tables)
```

```
  File "/tmp/ipython-input-311283708.py", line 32
    print("Tables in the database:", tables)tables = [('users',)]
                                            ^
SyntaxError: invalid syntax
```

```
cur.execute("""
CREATE TABLE IF NOT EXISTS properties (
  property_id INTEGER PRIMARY KEY,
  location TEXT NOT NULL,
  type TEXT NOT NULL,
  price_per_night REAL NOT NULL,
  tags TEXT
)
""")
# Commit any pending changes so chema is updated
conn.commit()
#Check all tables in the datanase
cur.execute("SELECT name FROM sqlite_master WHERE type='table'")
tables = cur.fetchall()
print("Tables in the database:", tables)

# Insert sample data
cur.execute("DELETE FROM users") # Clear existing data
for u in users:
    cur.execute("INSERT INTO users VALUES (?, ?, ?, ?, ?)", (
        u.user_id, u.name, u.group_size, u.preferred_environment, u.budget
    ))
print_table(cur, "Users table after initial inserts:", table_name="users")

for p in properties:
  cur.execute("INSERT INTO properties VALUES (?, ?, ?, ?, ?)", (
      p.property_id, p.location, p.type, p.price_per_night,
      ",".join(p.features), ",".join(p.tags)
  ))
print_table(cur, "Properties table after initial inserts:", table_name="properties")
```

双击（或按回车键）即可修改

```
# Query: all properties under a given user's budget

target_user_id = 102,
cur.execute("SELECT location, type, price_per_night FROM properties WHERE price_per_night <= ?")
for row in cur.fetchall():
    print(f"{row[1].title()} in {row[0]} (${row[2]}/night)")
conn.close()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-3284671218.py in <cell line: 0>()
      2
      3 target_user_id = 102,
----> 4 cur.execute("SELECT location, type, price_per_night FROM properties WHERE price_per_night <= ?")
      5 for row in cur.fetchall():
      6     print(f"{row[1].title()} in {row[0]} (${row[2]}/night)")

NameError: name 'cur' is not defined
```