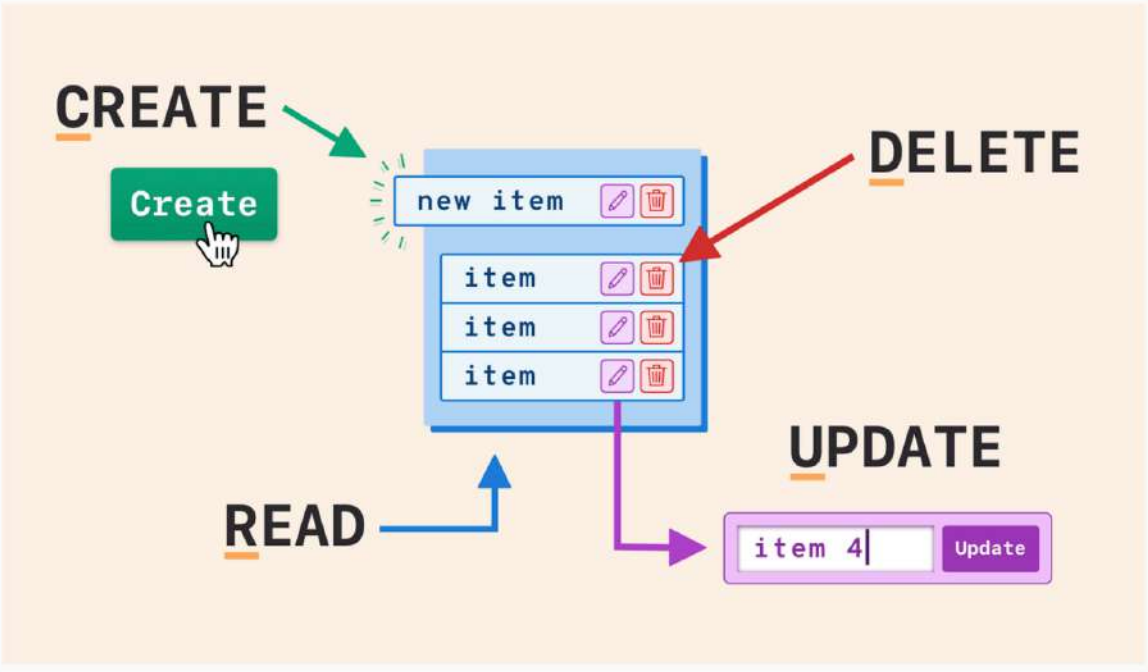
CRUD app in Python



Introduction to CRUD in Python

🔷 What is CRUD?

CRUD stands for the four basic operations we can perform on data:

- **C → Create**
- **R → Read**
- **U → Update**
- **D → Delete**

These four actions are the foundation of any database or file-based system.

🔷 Importance of CRUD

1. Used in every application that stores and manages data.
2. Allows adding, viewing, editing, and removing data easily.
3. Forms the base of database systems, web apps, and APIs.
4. Helps maintain data accuracy and user control.

🔷 CRUD in Python

In Python, CRUD operations can be implemented using:

- **File handling** (for small projects)
- **Databases** (like MySQL, SQLite, etc.)

This note explains a simple **file-based CRUD system** that works with a text file (for example, `cities.txt`).

🔷 Tools Used

- **File I/O** – to read and write data
- **List** – to store multiple records in memory
- **Functions** – to organize code for each operation
- **Menu-driven program** – to provide user options

🗃️ Example File:

Each city name is stored in a text file:

```
Delhi Mumbai Kolkata Chennai
```

🧠 Program Goal

To build a small Python program that can:

- Add new city names
- Display all stored cities
- Edit existing names
- Delete unwanted entries

🔷 Meaning:

"Create" means **adding new data** into the file.
In our program, it means adding a **new city name** to `cities.txt`.

🔷 Steps to Create Record

1. Open the file in **append mode ('a')**
2. Accept a new record from the user
3. Write it to the file followed by `\n`
4. Handle any possible errors

🔷 Code Example:

```
def create_record(new_record):     try:
with open(FILE_NAME, 'a') as file:             if
new_record:                 file.write(new_record
+ '\n')     except:         print("ERROR: Couldn't
create a new record.")
```

🔷 Explanation:

- `with open(FILE_NAME, 'a')`: **opens file for appending, keeping old data safe**
- `if new_record`: **ensures the user doesn't enter blank data**
- `file.write(new_record + '\n')`: **adds the new data in a new line**

🔷 Output Example:

```
Enter city name: Pune Record created successfully.
```

🔷 Key Points:

- Always use **append mode** for Create.
- Use **error handling** to avoid crashes.
- Validate input before saving.

Read Operation

🔷 Meaning:

"Read" means **displaying or retrieving** the stored data from the file.

🔷 Steps to Read Records:

1. Open the file in **read mode ('r')**
2. Read all lines and remove extra spaces
3. Display each record with an index number
4. Handle the case when the file is empty

🔷 Code Example:

```
def display_records():     records = read_all_records()     if not records:
print("\n--- Records not found ---")
return records      print("\n----- Current Records -----")     for index, record in enumerate(records):         print(index + 1, " ", record)     print("------------------------------")     return records
```

🔷 Helper Function:

```
def read_all_records():     try:         with open(FILE_NAME, 'r') as file:             records = [line.strip() for line in file if line.strip()]
return records     except:         return []
```

🔷 Output Example:

```
----- Current Records ----- 1  Delhi 2  Mumbai 3  Kolkata 4  Chennai --------------------------
```

🔷 Key Points:

- `readlines()` **or list comprehension can be used.**
- `strip()` **removes** `\n`.
- **Always check if the list is empty before displaying.**

Update Operation

🔷 Meaning:

"Update" means **modifying an existing record** in the file.
In our app, it allows the user to **change a city name.**

🔷 Steps to Update Record:

1. Read all records into a list
2. Ask user for record number to update
3. Replace the old value with the new one
4. Rewrite all records to the file

🔷 Code Example:

```
def update_record(index, new_item):     records = read_all_records()     idx = index - 1     if 0 <= idx < len(records):         records[idx] = new_item         write_all_records(records)         print("Record updated successfully")     else:         print("Error: Invalid record number.")
```

🔷 Helper Function to Write:

```
def write_all_records(records):     with open(FILE_NAME, 'w') as file:         for record in records:             file.write(record + '\n')
```

🔷 Output Example:

```
----- Current Records ----- 1  Delhi 2  Mumbai 3 Kolkata 4  Chennai ---------------------------
Enter record number to update: 2 Enter new city name: Bengaluru Record updated successfully.
```

🔷 Key Points:

- Index numbers start from 1 for the user.
- Internally, convert to 0-based (`index-1`).
- Use `'w'` mode to **overwrite** the updated list.

Delete Operation & Conclusion

🔷 Meaning:

"Delete" means **removing unwanted data** from the file.

🔷 Steps to Delete Record:

1. Read all records into a list
2. Ask for the record number to delete
3. Remove that item using `pop()`
4. Rewrite the updated list into the file

🔷 Code Example:

```
def delete_record(index):     records = read_all_records()
idx = index - 1     if 0 <= idx < len(records):
records.pop(idx)         write_all_records(records)
print("Record deleted successfully")     else:
print("ERROR: delete error.")
```

🔷 Output Example:

```
----- Current Records ----- 1  Delhi 2  Mumbai 3  Kolkata
4  Chennai -------------------------- Enter record
number: 3 Record deleted successfully.
```

🔷 Key Points:

- Use `pop()` to remove by index.
- Always check index validity before deletion.
- Rewrite all records to update the file.

🏁 Conclusion

The **File-based CRUD Application** is a simple way to learn:

- File reading and writing
- Exception handling
- Menu-driven logic
- Data management in Python

Such programs are the **foundation of database-driven systems** and help in understanding how larger CRUD systems (like in MySQL or Django) actually work behind the scenes.