Project Title: Email Slicer using python

<u>Introduction to the Email Slicer :</u>

The Advanced Email Slicer is a Python-based utility that takes one or more email addresses as input and intelligently extracts key components:
Username (the part before the @)
Domain name (excluding the TLD)
TLD (Top-Level Domain, like .com, .org, etc.)

Unlike basic implementations, this slicer includes:

Email format validation using regular expressions
Support for multiple emails entered at once
Error handling for malformed inputs
Optional result saving to a text file for record-keeping or further analysis

This tool is especially useful for:

Educational projects (learning about string manipulation and regex)
Small-scale data parsing tasks
Preprocessing emails for mailing lists, analytics, or admin tools.
It is designed to be run from the command line and offers a clear and user-friendly experience for both casual users and developers.

🔑 **<u>Key Features:</u>**

✅ 1. Email Validation:
Uses regular expressions to verify if an input is a valid email address.
Detects and rejects malformed or incomplete addresses.

✅ 2. Multiple Email Handling:
Accepts multiple email addresses in a single input (comma-separated).
Processes each email individually with clear output per entry.

✅ 3. Detailed Email Breakdown:
Slices each email into:
Username – The part before the @

Domain name – The main part of the address after the @ (excluding TLD)
TLD (Top-Level Domain) – Such as .com, .org, .net

✅ 4. Whitespace & Edge Case Handling:
Strips leading/trailing spaces.
Ignores blank or invalid entries cleanly.

✅ 5. Formatted Output:
Neatly displays results for each email.
Uses a structured and user-friendly format.

✅ 6. Save to File (Optional):
Allows users to save results to a .txt file.
Handy for keeping logs or exporting data.

✅ 7. Extensible Design:
Clean, modular functions make it easy to:
Add GUI or web support
Integrate into larger applications
Expand functionality (e.g., email provider detection)

🛠️ Technologies Used:

💻 Programming Language
Python 3.x – The core language used for building the entire application.

Built-in Python Library:

re – For regular expression matching, used to validate and parse email addresses.

🎯 Importance of the Advanced Email Slicer in Python:

The Advanced Email Slicer serves as a practical and educational tool for understanding how to manipulate and validate structured user input, such as email addresses. Here's why it's important:
🔐 1. Data Validation & Integrity:

Helps ensure that only properly formatted email addresses are processed.
Prevents errors in applications that depend on reliable contact information.

🧠 2. String Manipulation Practice:
Offers hands-on experience with string methods, slicing, and parsing techniques.
Demonstrates how to clean and extract meaningful data from raw text inputs.

🧪 3. Real-World Application:
Used in form validation, signup systems, email marketing tools, and CRM software.
Acts as a foundational component in larger systems that rely on email-based communication.

💼 4. Useful for Developers & Analysts:
Enables quick analysis of email data (e.g., identifying common domains or user patterns).
Can be extended to categorize users or identify organization-specific domains.

🧰 5. Scalable and Extensible:
The modular code structure makes it easy to integrate into GUI apps, web apps (Flask/Django), or automation scripts.
Easily adaptable to support CSV import/export or bulk email operations.

📚 6. Educational Value:
Great for teaching or learning about:
Regular expressions (re)
Error handling
File I/O operations
Code structuring with functions

📄 **Summary**:

The **Advanced Email Slicer** is a Python-based command-line tool designed to parse and analyze one or more email addresses. Unlike basic slicers, this advanced version performs **email validation**, handles **multiple entries**, and accurately extracts three key components from each email:

- **Username** (before the `@`)

- **Domain name** (after the `@`, excluding the TLD)
- **TLD** (Top-Level Domain, such as `.com`, `.org`)

It uses **regular expressions** for format checking, cleans whitespace, gracefully skips invalid inputs, and optionally **saves the output to a text file**. With a modular design and clean structure, the script is both beginner-friendly and highly extensible—making it suitable for learning, real-world preprocessing, or integration into larger projects.

This project demonstrates essential Python skills including:

- String manipulation
- Regular expressions
- File handling
- Input validation
- Functional programming practices

📌 Scope of the Advanced Email Slicer in Python:

The Advanced Email Slicer project is designed to provide a functional and educational tool for parsing and analyzing email addresses. Its scope covers both practical use cases and academic learning objectives within the domain of string processing and data validation.

🔍 In-Scope Features:

1. Email Validation:

Uses regular expressions to detect and reject improperly formatted email addresses.

2. Multiple Email Processing:

Accepts a list of comma-separated emails and handles them individually.

3. Data Extraction:

Slices valid emails into:

Username

Domain name (excluding the TLD)

TLD (e.g., .com, .org)

4. User Interaction via CLI:

Clean, user-friendly command-line prompts and formatted output.

5. File Output (Optional):

Allows users to save the extracted data toa .txt file for further use.

6. Modular & Reusable Code:

Designed for easy enhancement or integration into larger applications (web apps, GUIs, or automation scripts).

🧩 Future Scope & Extensions:

Integration with a GUI (e.g., Tkinter, PyQt) or web interface (e.g., Flask)

Domain reputation or MX record checking

Bulk CSV import/export for enterprise-level email processing

Categorization based on common email providers (Gmail, Outlook, etc)

Code:

```
import re

def validate_email(email):

    """Validates the email format."""

    pattern = r"^[\w\.-]+@[\w\.-]+\.\w{2,}$"

    return re.match(pattern, email.strip())

def slice_email(email):

    """Extracts username, domain, and TLD from email."""

    email = email.strip()

    try:

        username, domain = email.split("@")
        if "." in domain:
            domain_name, tld = domain.rsplit(".", 1)
        else:
            domain_name, tld = domain, ""
        return {
            "Email": email,
            "Username": username,
            "Domain": domain_name,
            "TLD": tld
        }
    except ValueError:
        return None
def process_emails(email_list):
    """Processes multiple emails."""
    results = []
    for email in email_list:
        if validate_email(email):
            data = slice_email(email)
            results.append(data)
        else:
            print(f"❌ Invalid email format: {email}")
    return results
def display_results(results):
    for entry in results:
        print("\n✅ Email Sliced:")

        for key, value in entry.items():

            print(f"{key}: {value}")

def save_results(results, filename="email_slices.txt"):

    with open(filename, "w") as f:

        for entry in results:

            f.write(f"{entry['Email']} -> Username: {entry['Username']}, Domain: {entry['Domain']}, TLD: {entry['TLD']}\n")

    print(f"\n📁 Results saved to {filename}")

# --- MAIN ---

if __name__ == "__main__":

    raw_input = input("Enter one or more email addresses (comma-separated):\n")

    email_list = raw_input.split(",")

    sliced = process_emails(email_list)

    display_results(sliced)

    save_option = input("\nDo you want to save the results to a file? (y/n): ").lower()

    if save_option == 'y':

        save_results(sliced)
```

✅ Sample Run:

Input:

```
Enter one or more email addresses (comma-separated):

john.doe@example.com, user123@sub.domain.org, fakeemail@, test@domain
```

Output:

```
✅ Email Sliced:
```

```
Email: john.doe@example.com

Username: john.doe

Domain: example

TLD: com

✅ Email Sliced:

Email: user123@sub.domain.org

Username: user123

Domain: sub.domain

TLD: org

❌ Invalid email format: fakeemail@

❌ Invalid email format: test@domain
```

✅ Conclusion:

The Advanced Email Slicer in Python effectively demonstrates how to validate and process structured user input through string manipulation, regular expressions, and clean program design. By slicing emails into their core components—username, domain, and TLD—the tool provides both a practical utility and a strong learning foundation for developers.

This project goes beyond basic functionality by handling multiple emails, managing invalid formats, and offering optional file output, all while maintaining a lightweight and user-friendly interface. It highlights essential Python skills such as input handling, data validation, modular function design, and file operations.

In conclusion, the Advanced Email Slicer is a valuable tool for educational use, data preprocessing tasks, and can be easily extended into more complex systems such as web applications, email analytics tools, or user data processing pipelines.