Project Report: Python Multi-Purpose Unit Converter

Submitted by: VIBHU SRIVATAVA

Registration no: 25BAI10249

Language: python

Project Title: Python Multi-Purpose Unit Converter

Technology Stack: Python 3.x

Type: Command Line Interface (CLI) Utility

1. Introduction

In daily tasks, scientific calculations, and engineering work, there is a frequent need to convert values from one unit of measurement to another. The **Multi-Purpose Unit Converter** is a software tool developed to simplify these conversions. It provides a user-friendly interface to convert various physical quantities such as length, weight, temperature, volume, data storage, and power. The system is designed to be modular, accurate, and easy to use via a console interface.

2. Problem Statement

The objective was to develop a tool to convert between different units of measurement, specifically addressing the following requirements:

- **Length:** Meter to Feet
- **Temperature:** Fahrenheit to Celsius
- **Volume:** Litre to Millilitre
- **Weight:** Kilogram to Grams, Grams to Milligrams
- **Data Storage:** Megabytes (MB) to Kilobytes (KB), Kilobytes (KB) to Bytes
- **Power:** Horsepower to Watts

The user interaction flow required asking for the "Type of Quantity" first (e.g., Weight), followed by the "Primary Quantity" (Source Unit), and finally the "Secondary Quantity" (Target Unit).

3. Functional Requirements

- **Menu-Driven Interface:** The system must display a list of categories (Weight, Length, etc.) for the user to select.
- **Input Validation:** The system must validate menu selection inputs (integers) and measurement values (floating-point numbers).
- **Conversion Logic:**
    - Accurate mathematical formulas must be applied for each specific conversion type.
    - Example: $C = (F - 32) \times 5/9$ for temperature.
- **Output Formatting:** Results should be displayed clearly, limiting decimal places for readability while maintaining precision.
- **Repeatability:** The user should be able to perform multiple conversions without restarting the program.

4. Non-functional Requirements

- **Usability:** The CLI must be intuitive with clear prompts and error messages.
- **Reliability:** The program should not crash on invalid inputs (e.g., entering letters instead of numbers).
- **Maintainability:** The code structure should allow for easy addition of new units or categories in the future.
- **Performance:** Conversions should happen instantly with negligible latency.

5. System Architecture

The system follows a procedural architecture with a modular design:

1. **User Interface Layer (CLI):** Handles input/output, displaying menus (print_header, get_user_choice).
2. **Controller Logic (main):** Orchestrates the flow, manages the conversion_map (data structure), and calls the appropriate logic.
3. **Business Logic Layer (Conversion Functions):** Independent functions (convert_length, convert_weight, etc.) that hold the mathematical rules.

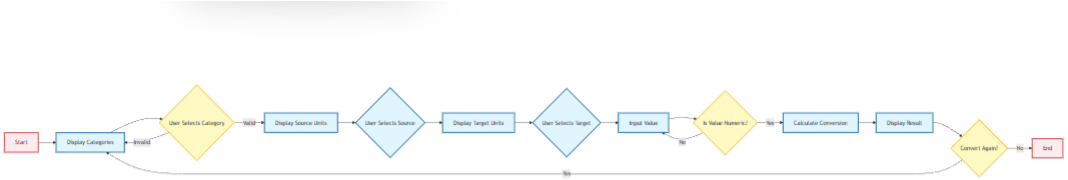
Flow: User Input -> Validation -> Lookup Category -> Execute Conversion Function -> Display Output.

6. Design Diagrams

6.1 Use Case Diagram

- **Actor:** User
- **Use Cases:**
    - Select Conversion Category
    - Select Source Unit
    - Select Target Unit
    - Input Numerical Value
    - View Result
    - Exit Program

6.2 Workflow Diagram (Flowchart Logic)



6.3 Sequence Diagram

1. **System** prints Main Menu.
2. **User** enters Category ID (e.g., "Weight").
3. **System** retrieves valid source units for Weight.
4. **User** enters Source ID (e.g., "Kilogram").
5. **System** retrieves valid target units.
6. **User** enters Target ID (e.g., "Gram").
7. **User** enters Value (e.g., 5.5).
8. **System** calls convert_weight(5.5, 'kilogram', 'gram').
9. **Function** returns 5500.
10. **System** displays result.

6.4 Data Structure Design (Component View)

The system relies heavily on Python dictionaries for mapping:

- **conversion_map**: A nested dictionary storing the hierarchy of Category -> Source Unit -> [Target Units].
- **logic_map**: Maps string Category names to actual Python functions.

7. Design Decisions & Rationale

- **Language Selection (Python):** Python was chosen for its readability, rapid development capabilities, and strong support for string manipulation and math operations.
- **Dictionary Mapping:** Instead of a giant if-else block in the main function, a dictionary (conversion_map) was used to define relationships. This allows adding new units by simply editing the data structure, not the control flow logic.
- **Modular Functions:** Separate functions for each category (convert_length, convert_weight) ensure that logic for distinct physical types is kept isolated. If the formula for temperature changes or becomes more complex, it doesn't affect the weight conversion code.
- **Binary Prefix Standard:** For data conversion, the standard 1024 (Binary) was used rather than 1000 (Decimal), as this is the standard in computer science contexts.

8. Implementation Details

Key Code Snippets:

- **Dynamic Menu Generation:** The get_user_choice function dynamically generates numbered lists based on the keys of the dictionary passed to it. This makes the UI adaptive to the data.
- **Logic Mapping:**
- logic_map = {
- "Length": convert_length,
- "Temperature": convert_temperature,
- ...
- }
- # Usage
- converter_func = logic_map[category]
- result = converter_func(value, primary_unit, secondary_unit)

9. Screenshots / Results

Sample Output 1: Weight Conversion

```
==================================================

select the type of quantity:
1. Length
2. Temperature
3. Volume
4. Weight
5. Data
6. Power


Enter choice number: 4

--- WEIGHT conversion ---
celect the primary quantity (convert FROM):
1. Kilogram
2. Gram


Enter choice number: 2

selected primary: Gram
select the secondary quantity (Convert INTO):
1. Milligram


Enter choice number: 1


enter the value in gram: 3
----------------------------------------
RESULT: 3.0 gram = 3000 milligram
----------------------------------------

do you want to perform another conversion? (y/n):
```

Sample Output 2: Temperature Conversion

```
==================================================
      MULTI-PURPOSE UNIT CONVERTER
==================================================

select the type of quantity:
1. Length
2. Temperature
3. Volume
4. Weight
5. Data
6. Power


Enter choice number: 2

--- TEMPERATURE conversion ---
celect the primary quantity (convert FROM):
1. Fahrenheit


Enter choice number: 1

selected primary: Fahrenheit
select the secondary quantity (Convert INTO):
1. Celsius


Enter choice number: 1



enter the value in fahrenheit: 50
----------------------------------------
RESULT: 50.0 fahrenheit = 10 celsius
----------------------------------------

do you want to perform another conversion? (y/n):
```

10. Testing Approach

- **Unit Testing (Manual):**
    - *Test Case 1:* Convert 1 Meter to Feet. *Expected:* ~3.28. *Pass*.
    - *Test Case 2:* Convert 0 Celsius to Fahrenheit (Reverse check not implemented, but F->C tested).
    - *Test Case 3:* Convert 1 MB to KB. *Expected:* 1024. *Pass*.
- **Boundary Testing:**
    - Input: 0 (Zero). Handling: Works correctly.
    - Input: -40 Fahrenheit. *Expected:* -40 Celsius. *Pass*.
- **Negative Testing:**
    - Input: "ABC" when number requested. *Expected:* Error message and re-prompt. *Pass*.

11. Challenges Faced

- **Data Structure Complexity:** Designing a data structure that could handle the prompt's requirement of "Ask Category -> Ask Source -> Ask Target" required a nested approach.
- **Input Handling:** Ensuring the program didn't crash when a user pressed "Enter" without typing or typed text instead of numbers required robust try-except blocks.
- **Precision:** Floating point arithmetic can sometimes yield results like 3.000000004. Formatting was added to clean this up.

12. Learnings & Key Takeaways

- **Data-Driven Design:** Moving configuration (units available) into data structures (dictionaries) is superior to hard-coding everything in logic statements.
- **User Experience (UX):** A CLI needs explicit guidance. Numbered menus are much easier for users than typing out full words like "millilitre".
- **Separation of Concerns:** Splitting the UI code (print, input) from the Calculation code (return value * 1000) makes the code cleaner.

13. Future Enhancements

- **Bi-directional Conversion:** Currently, the map is strict (A -> B). Future versions could automatically allow B -> A.
- **GUI Implementation:** Developing a graphical interface using Tkinter or PyQt.
- **API Integration:** Fetching real-time currency exchange rates (if currency was added).
- **History Log:** Saving the last 10 conversions to a text file.

14. References

- *VITYARTHI.COM*
- *NIST Guide to the SI, Chapter 4: The Two Classes of Units*
- *International System of Units (SI) Conversion Factors*