Phase 1 Tasks

Created by	Ⓜ Muhammad Yousuf
Date	@August 13, 2025

1. Variables & Data Types

- Create a program that asks a user for their **birthdate** (`YYYY-MM-DD`) and calculates their exact **age in years, months, and days**.

- Design a currency converter that works **without using any APIs. U**se a hardcoded data structure of conversion rates but allow the user to switch the base (from) currency dynamically. Input 2 currencies from given options (From and To) and value of base currency then display result after conversion

---

2. Strings

- Write a program that finds the **most frequent word** in a given paragraph, ignoring punctuation and case.

- Implement a function to check whether a given string is a **valid mathematical expression** (only digits, `+` ,- ,* , `/` , and parentheses) and whether the parentheses are balanced.

---

3. Conditional Statements

- Build a grading system that calculates grades not only from marks but also from **attendance percentage** and gives warnings for low attendance.

- Create a simple **rule-based chatbot** that responds differently based on user input keywords, using only `if/elif/else` .

---

4. Lists

- Take a list of mixed integers from user and remove duplicates, but **keep only the last occurrence** of each element. Then show final output

- Implement a "chunk splitter". Given a list and a chunk size `n`, split it into sublists of exactly `n` elements, padding with `None` if needed.

---

5. Tuples

- Create a program that generates a list of tuples where each tuple contains a number and its cube, for numbers from 1 to 20.

- Implement a function that takes a tuple of tuples and **sorts them by the second element in descending order**, breaking ties with the first element ascending.

---

6. Dictionary

- Count character frequencies in a string and store them in a dictionary, sorted alphabetically by character.

You are tasked with simulating a shopping cart system. Given is

- **Product Catalog:** A dictionary containing `{product_name: price_per_unit}`.
- **Cart:** A dictionary containing `{product_name: quantity}`.
- **Discounts:** A dictionary containing `{product_name: discount_percentage}` (0 to 100), representing percentage discounts on each product.

Your Tasks:

1. Build a new dictionary called `final_bill` where each key is the product name and each value is the **total cost after applying the discount** for that product.
2. Display the `final_bill` dictionary in the terminal.

Rules:

- If a product in the cart has no discount in the discounts dictionary, treat its discount as `0%`.
- All calculations should round to **2 decimal places**.

Example Input:

```
product_catalog = {
    "apple": 3.0,
    "banana": 1.5,
    "milk": 2.5,
    "bread": 2.0
}
```

```
cart = {
   "apple": 4,
   "banana": 6,
   "milk": 2,
   "bread": 3
}

discounts = {
   "apple": 10,   # 10% off
   "milk": 20     # 20% off
   # if no discount is there for the product, it means discount = 0%
}
```

Expected Output:

```
{'apple': 10.8, 'banana': 9.0, 'milk': 4.0, 'bread': 6.0}
```

---

7. Set

- Given two lists, find the **elements that appear in exactly one list**, without using loops (just set operations).

- Build a program to find the **smallest missing positive integer** from a list using set operations.

---

8. Loops

- Implement a number guessing game where the program selects a random number and the user gets **limited attempts**. Less tries means more scores. Initially set score per attempt and limit.
  Example:
  - Limit = 10 and limitScore = 10, means max score = 100
  - user tried once and it was correct, score should be 100
  - user failed on first attempt, tried second and it was correct, score should be 90
  - and so on...
  - At last attempt the game should end with game over and 0 scores

- Write a program that finds all **prime numbers within a range** using the **Sieve of Eratosthenes** algorithm.

Steps

Let's find all primes ≤ **30**.

1. **Make a list of numbers from 2 to n (input n from user)**

   ```
   [2, 3, 4, 5, 6, 7, 8, 9, ..., 30]
   ```

   We don't include 0 or 1 because they are not primes.

2. **Start with the first prime number: 2**
   - Mark all multiples of 2 as **not prime** (except 2 itself).

     Cross out: 4, 6, 8, 10, 12, ..., 30.

3. **Move to the next number that's not crossed out: 3**
   - Mark all multiples of 3 as **not prime** (except 3).

     Cross out: 6 (already crossed), 9, 12, 15, ..., 30.

4. **Next uncrossed number: 5**
   - Mark all multiples of 5 as **not prime**. (except 5 itself)

     Cross out: 10, 15, 20, 25, 30.

5. **Continue** until you reach `√n` (square root of n).
   - Why stop at `√n`? Because any composite number ≤ n must have a factor ≤ √n

6. **The remaining uncrossed numbers are all primes.**
7. Store them in a list and display that list

---

9. Functions

- Create a function that takes a list of numbers and returns the **mean, median, and mode**.

- Implement a function that takes two sorted lists and **merges them into one sorted list** without using `sort()` or `sorted()`.

---

10. Recursion

- Write a recursive function to **reverse a string**.

- Write a recursive function to generate **all permutations** of a given list.

What Is a Permutation?