Final Assignment

Problem 1: Simple & Compound Interest Calculator

Description: Create a versatile interest calculator. The program should first ask the user whether they want to calculate 'simple' or 'compound' interest.

- If they choose 'simple', it should ask for the Principal amount (P), annual rate of interest (R), and time period in years (T). It should then calculate and display the Simple Interest and the total amount.
- If they choose 'compound', it should ask for the Principal amount (P), annual rate of interest (R), the number of times interest is compounded per year (n), and the time period in years (T). It should then calculate and display the Compound Interest and the total amount. Handle the case where the user enters an invalid choice.

Formulae:

- Simple Interest: SI=fracPtimesRtimesT100
- Total Amount (Simple): A=P+SI
- Total Amount (Compound): A=Pleft(1+fracR/100nright)nt
- Compound Interest: CI=A−P

Hint: Start by taking the user's choice for the type of interest. Use an `if` statement to check if the input string is "simple". Use an `elif` for "compound". Remember to convert the input strings for numerical values to `float` or `int` before performing calculations.

Problem 2: Loan Eligibility Checker

Description: A bank has a set of rules to determine if a customer is eligible for a personal loan. Write a program that checks eligibility based on the following criteria:

- Minimum annual income: ₹5,00,000
- Minimum credit score: 650
- Minimum age: 21 years

The program should ask the user for their annual income, credit score, and age. It should then use nested conditional statements to decide and print one of the following outputs:

- "Congratulations! You are eligible for the loan."
- "Sorry, you are not eligible. Your income does not meet the criteria."
- "Sorry, you are not eligible. Your credit score is too low."
- "Sorry, you are not eligible due to your age."

Hint: It's best to check the criteria in a sequence. You can use a main `if` statement to check the most important criterion first (e.g., income). Inside that, you can nest another `if` to check the next criterion (e.g., credit score), and so on.

Problem 3: EMI Amortization Schedule Generator

Description: Create a function that calculates and prints a monthly amortization schedule for a loan. The program should take the loan amount (Principal), the annual interest rate, and the loan term (in years) as input from the user.

First, calculate the Equated Monthly Installment (EMI) using the formula below. Then, use a loop to print a month-by-month breakdown showing the month number, the EMI paid, the interest paid for that month, the principal paid for that month, and the remaining balance.

EMI Formula: EMI=Ptimesrtimesfrac(1+r)^{n}(1+r)$^{n-1}$ where:

- P = Principal loan amount
- r = Monthly interest rate (R/(12times100))
- n = Total number of installments (Loan term in years times12)

Hint: Inside the `for` loop, for each month, you first need to calculate the interest component for that month (Remaining Balance times monthly interest rate). The principal component is then `EMI - Interest Component`. Finally, update the remaining balance by subtracting the principal component.

Problem 4: Bank Transaction Ledger

Description: Simulate a simple bank transaction ledger. The program should start by asking for an initial balance. Then, it should allow the user to enter a series of transactions (deposits and withdrawals).

- Deposits should be entered as positive numbers (e.g., `5000`).
- Withdrawals should be entered as negative numbers (e.g., `-1500`).

The user can enter transactions one by one. After each entry, the program should ask if they want to add another transaction. When the user is done, the program should:

1. Print the list of all transactions.
2. Calculate and print the total number of deposits and withdrawals.
3. Calculate and print the final account balance.

Hint: Initialize an empty list, say `transactions = []`. Use a `while True` loop to ask for input. Inside the loop, get the transaction value and `append()` it to the list. Ask the user if they want to continue; if they say 'no', `break` the loop. After the loop, iterate through the `transactions` list with a `for` loop to perform your calculations.

Problem 5: Investment Portfolio Return Calculator

Description: Write a program to calculate the total return on a simple investment portfolio. The portfolio data will be stored in a list of lists (a nested list), where each inner list contains information about a single stock: `[stock_name, number_of_shares, purchase_price_per_share, current_price_per_share]`.

Create a function that takes this portfolio list as an argument. The function should iterate through each stock in the portfolio and calculate:

- The total initial investment for that stock.
- The current market value for that stock.
- The profit or loss for that stock.

Finally, the program should print a summary table for each stock and then display the total portfolio initial investment, total current market value, and the overall profit/loss for the entire portfolio.

Example Portfolio Data:

```
portfolio = [
   ['RELIANCE', 50, 2500, 2800],
   ['TCS', 100, 3200, 3550],
   ['HDFCBANK', 150, 1400, 1650]
]
```

Hint: Inside your `for` loop, you'll process one inner list at a time (e.g., `['RELIANCE', 50, 2500, 2800]`). You can access the number of shares with `stock[1]`, purchase price with `stock[2]`, and so on. Keep track of total investment and total current value in variables that you update with each iteration of the loop.