CHAPTER 1

Description

The goal of this project is to create an algorithm that will allow FarziBank to mimic 20,000 clients, their account information, and activity over the course of a year. Customers can be identified by their ID, gender, age, marital status, number of children, degree of education, number of accounts, annual income, and total credit line, among other factors. Account numbers, dates of opening, credit lines, annual fees, and APR are among the aspects of the accounts. Purchase amount, cash advance amount, balance, payment amount, and several other factors are also included in account activity.

The algorithm is structured in the following steps:

1. **Generating Customers** :

Generate a 7-digit unique Customer ID for each customer in Step 1 of creating customers. Age, gender, marital status, number of children, education level, annual income, number of accounts, and total credit line are all assigned at random.

2. **Creating Accounts** :

- Choose each account's opening date at random (before to January 1, 2022).
- Based on regulations, generate an 8-digit unique account number.
- Define the account's credit line in relation to the customer's overall credit line.
- Determine the annual charge and choose a random interest rate.

3. **Generating Account Activity :**

- Simulate activity for each account for a full year, from January 1 to December 31, 2022.
- Calculate the number of times you will use your card (for purchases or cash advances) every 'd' days.
- Calculate a purchase or cash advance amount using the credit or cash on hand.
- Calculate the balance's daily interest rate.
- Based on consumer behavior, simulate various payment scenarios.

4. **Monthly Calculations**

- At the conclusion of every month, figure out the closing balance, the bare minimum payment, the total amount of purchases, cash advances, payments, and interest charges.

5. **Delinquency and late payments**:

- If the client has not made the bare minimum payment by the deadline:
  - Define the amount past due and include a late payment cost.
  - Increase the delinquent tally.

6. **Account Closure** :

- If the delinquency counter reaches 3, close the account.

This algorithm efficiently generates and simulates customer and account data, ensuring accuracy and adherence to the provided specifications. The resulting data can be used for further analysis by FarziBank.

.....

CHAPTER-2

2.1 Approach

The measures we'll take to generate 20,000 consumers with the required traits are as follows:

1. **Customer ID Generation**: Write a function called "generate_customer_id()" that generates a unique 7-digit customer ID at random for each customer.

2. **Age Generation** - We'll write a function called "generate_age()" to generate a random age between 20 and 80 for each customer.

3. **Gender Generation** - Write a function called "generate_gender()" to equally likely choose "Male" or "Female" at random for each customer.

4. **Marital Status Generation** - Build a function called "generate_marital_status(age)" that accepts age as an input and chooses a marital status at random from a distribution.

5. **Generation of the number of children**: Write a function called "generate_number_of_children(age)" that takes the age as an input and determines the number of children based on the stated probabilities.
6. **Generation of Education Levels** - Write a function called "generate_education_level(age)" that accepts age as an argument and generates education levels based on the stated probabilities.
7. **Annual Income Calculation**: Using the provided formula, develop a function called "generate_annual_income(education_level, age)" that computes the annual income.

8. **Number of Accounts Calculation** - Write a function that determines the number of accounts a client will have, generate_number_of_accounts(marital_status, number_of_children).
9. **Total Credit Line Calculation**: Write a function that determines a customer's total credit line, generate_total_credit_line(number_of_accounts, annual_income).
10. **Generate Customers**: - Using the functions described above, loop through 20,000 iterations to generate individual customers.
11. **Data Storage**: Each customer's data will be kept in a dictionary, and a list of all customers will be kept.
12. **Functionality and Testing**: At the end, a list of 20,000 clients is generated, each with their own special qualities.

We achieve modularity and maintainability in our programming by segmenting the issue into smaller functions. Because each function produces a particular characteristic, the code is simple to comprehend and debug. This strategy also gives for flexibility in case future revisions or feature requests arise.

2.2 Algorithm

2.2.1 Customer generation :

1. Define Probability Distributions
   - Define lists for age ranges, marital status probabilities, children probabilities, and education probabilities.
2. Define Functions for Generating Customer Attributes:
   - `**generate_customer_id**()`:
     - Generates a random 7-digit customer ID.
   - `**generate_age**()`:
     - Generates a random age between 20 and 80.
   - `**generate_gender**()`:
     - Randomly selects and returns 'Male' or 'Female'.
   - `**generate_marital_status**(age)`:
     - Takes age as input.
     - Uses defined age ranges and corresponding marital status probabilities to generate marital status.
     - Returns 'Single' or 'Married'.
   - `**generate_children**(age)`:
     - Takes age as input.
     - Uses defined age ranges and corresponding children probabilities to generate the number of children.
     - Returns a number between 0 and 4.
   - `**generate_education**(age)`:
     - Takes age as input.
     - Uses defined age ranges and corresponding education probabilities to generate education level.
     - Returns a number representing education level (0 for 'No Education' to 4 for 'Ph.D.').
   - `**generate_annual_income**(education_level, age)`:
     - Takes education level and age as input.
     - Calculates and returns the annual income based on the provided formula.
   - `**generate_num_accounts**(marital_status, num_children)`:

- Takes marital status and number of children as input.
- Calculates and returns the number of accounts based on the provided formula.

- `**generate_total_credit_line**(num_accounts, annual_income)`:
- Takes number of accounts and annual income as input.
- Calculates and returns the total credit line based on the provided formula.

3. **Generate 20,000 Customers**:
- Initialize an empty list `customers`.
- For each iteration (20,000 times):
- Generate attributes for a customer:
- Customer ID
- Age
- Gender
- Marital Status
- Number of Children
- Education Level
- Annual Income
- Number of Accounts
- Total Credit Line
- Append the generated customer data as a dictionary to the `customers` list.

4. **Output**:
- The `customers` list now contains 20,000 customer records, each with the specified characteristics.

This algorithm outlines the steps involved in generating customer data with the specified characteristics using the provided Python code.

2.2.2 Generating Accounts :

1. Define Constants
- Define the minimum and maximum interest rates.

2. Define Functions for Account Generation:
- `**generate_random_date**(age)`:
- Takes age as input.
- Calculates the earliest date based on the age and ensures it's not older than the customer's current age.
- Generates a random date before January 1st, 2022.
- `**generate_account_number**(customer_id, account_index)`:
- Takes Customer ID and account index as input.
- Generates an 8-digit account number based on the provided rule.
- `**generate_account_credit_line**(total_credit_line)`:
- Takes total credit line as input.
- Generates a random proportion of the customer's total credit line as the account credit line.
- `**generate_annual_fee**(account_credit_line)`:
- Takes account credit line as input.
- Calculates and returns the annual fee based on the provided formula.
- `**generate_annual_interest_rate**()`:
- Generates a random annual interest rate between 15% and 30%.

3. **Generate Account Information for Each Customer**:
- For each customer in the `customers` list:
- Initialize an empty list `accounts` to store account information.
- For each account index (from 1 to the number of accounts):

- Generate date opened using `generate_random_date(customer['Age'])`.
- Generate account number using `generate_account_number(customer['Customer ID'], account_index)`.
- Generate account credit line using `generate_account_credit_line(customer['Total Credit Line'])`.
- Generate annual fee using `generate_annual_fee(account_credit_line)`.
- Generate annual interest rate using `generate_annual_interest_rate()`.
- Create a dictionary `account_info` with the generated information.
- Append `account_info` to the `accounts` list.

- Assign `accounts` to the customer's record.

4. **Output**:
- Each customer's record in the `customers` list now includes account information.

This algorithm outlines the steps involved in generating accounts information for each customer based on the specified characteristics using the provided Python code.

2.2.3 Generating Account Activity

1. `**generate_random_days_between_uses**()`:
- Generate a random integer between 0 and 7 as `days_between_uses`.
- Return `days_between_uses`.
2. `**simulate_account_activity**(account)`:
- Initialize `today` to January 1, 2022.
- For each day in a year (365 iterations):
- Generate a random number of days between uses as `days_between_uses`.
- Add `days_between_uses` to `today`.
- If `today` is after January 1, 2023, exit the loop.
- Calculate `available_credit_line` as `CreditLimit - CurrentBalance`.
- Calculate `available_cash` as the minimum of 10% of `CreditLimit` and `available_credit_line`.
- Generate a random number (`rand_num`) between 0 and 1.
- If `rand_num` is less than 0.95 (95% chance of making a purchase):
- If `available_credit_line` is greater than 0:
- Generate a random `purchase_amount` between 0 and `available_credit_line`.
- Add `purchase_amount` to `CurrentBalance`.
- Else (5% chance of taking cash advance):
- If `available_cash` is greater than 0:
- Generate a random `cash_advance_amount` between 0 and `available_cash`.
- Add `cash_advance_amount` to `CurrentBalance`.
- Calculate `daily_interest_rate` as `AnnualInterestRate / 365`.
- Multiply `CurrentBalance` by `(1 + daily_interest_rate)`.
3. ‘**simulate_customer_payment**(account)`:
- Randomly select a payment scenario from ['exact', 'full_balance', 'proportion', 'payment_period'] as `payment_scenario`.
- Depending on `payment_scenario`:
- If `payment_scenario` is 'exact':
- Set `CurrentBalance` to 0.0.
- Else if `payment_scenario` is 'full_balance':
- Subtract `CurrentBalance` from itself.
- Else if `payment_scenario` is 'proportion':
- Generate a random proportion (`p`) between 0 and 1.
- Multiply `CurrentBalance` by `(1 - p)`.

- Else if `payment_scenario` is 'payment_period':
  - Generate a random number (`rand_num1`) between 0 and 1.
  - If `rand_num1` is less than 0.1 (10% pay entire balance within payment period):
    - Set `CurrentBalance` to 0.0.
  - Else if `rand_num1` is less than 0.15 (15% pay minimum amount due within payment period):
    - Subtract 10% of `CurrentBalance` from itself.

4. `**perform_monthly_calculations**(account)`:
   - Calculate `closing_balance` as `CurrentBalance`.
   - Calculate `minimum_amount_due` as 10% of `closing_balance`.
   - Calculate various monthly metrics:
     - `total_purchases`: Sum of purchase amounts.
     - `total_cash_advances`: Sum of cash advance amounts.
     - `total_payments`: Sum of payment amounts.
     - `total_interests`: Sum of interest amounts.
   - If the day of the month is greater than or equal to 11 and `PastDueAmount` is less than `minimum_amount_due`:
     - Set `PastDueAmount` to `minimum_amount_due`.
     - Add $30 to `CurrentBalance` (Late Payment Fee).
     - Increment `DelinquencyCounter` by 1.
   - If `DelinquencyCounter` is greater than 0 and `PastDueAmount` is 0:
     - Set `DelinquencyCounter` to 0.

5. **Main Simulation Loop**:
   - For each `customer` in `customers`:
     - For each `account` in `customer['Accounts']`:
       - Initialize `CurrentBalance`, `DelinquencyCounter`, and `PastDueAmount`.
       - Call `simulate_account_activity(account)`.
       - Call `simulate_customer_payment(account)`.
       - Call `perform_monthly_calculations(account)`.

.....

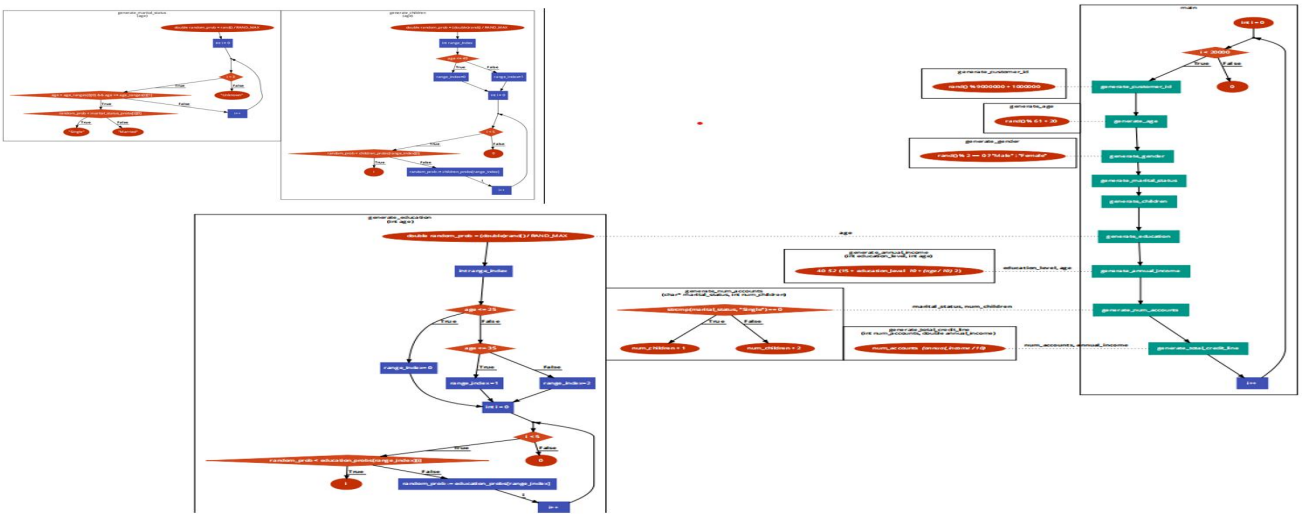
CHAPTER 3

Flowchart

3.1 Generate 20,000 customers

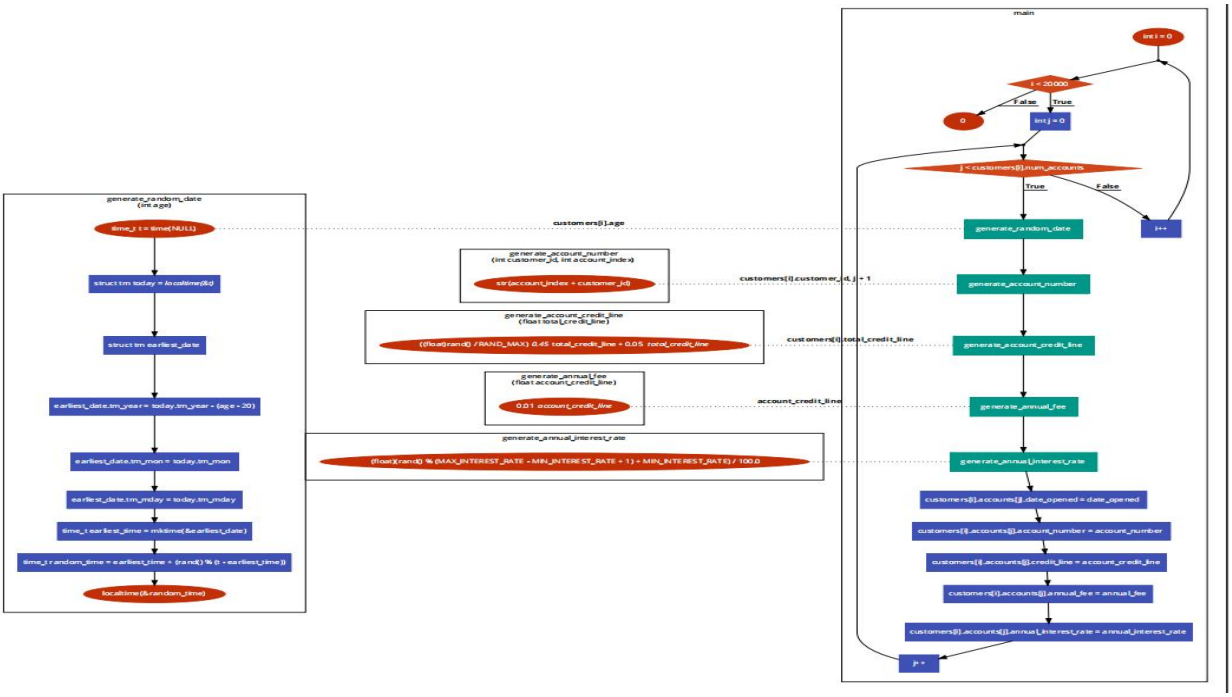
1. `**generate_customer_id**()`: Generates a random 7-digit customer ID.
2. `**generate_age**()`: Generates a random age between 20 and 80.
3. `**generate_gender**()`: Randomly selects and returns a gender from the options 'Male' or 'Female'.
4. `**generate_marital_status**(age)`: Based on the provided age, randomly determines the marital status ('Single' or 'Married') with specified probabilities.
5. `**generate_children**(age)`: Based on the provided age, randomly determines the number of children a person has, within predefined probabilities.
6. `**generate_education**(age)`: Based on the provided age, randomly assigns an education level represented as a numeric value.
7. `**generate_annual_income**(education_level, age)`: Calculates and returns the annual income based on education level and age according to a specific formula.
8. `**generate_num_accounts**(marital_status, num_children)`: Determines the number of accounts a person has based on marital status and number of children, following specified rules.
9. `**generate_total_credit_line**(num_accounts, annual_income)`: Computes the total credit line for a customer based on the number of accounts and annual income.

10. **Loop**: Generates 20,000 customer profiles using the methods described above and appends them to the `customers` list. The profiles include customer ID, age, gender, marital status, number of children, education level, annual income, number of accounts, and total credit line.



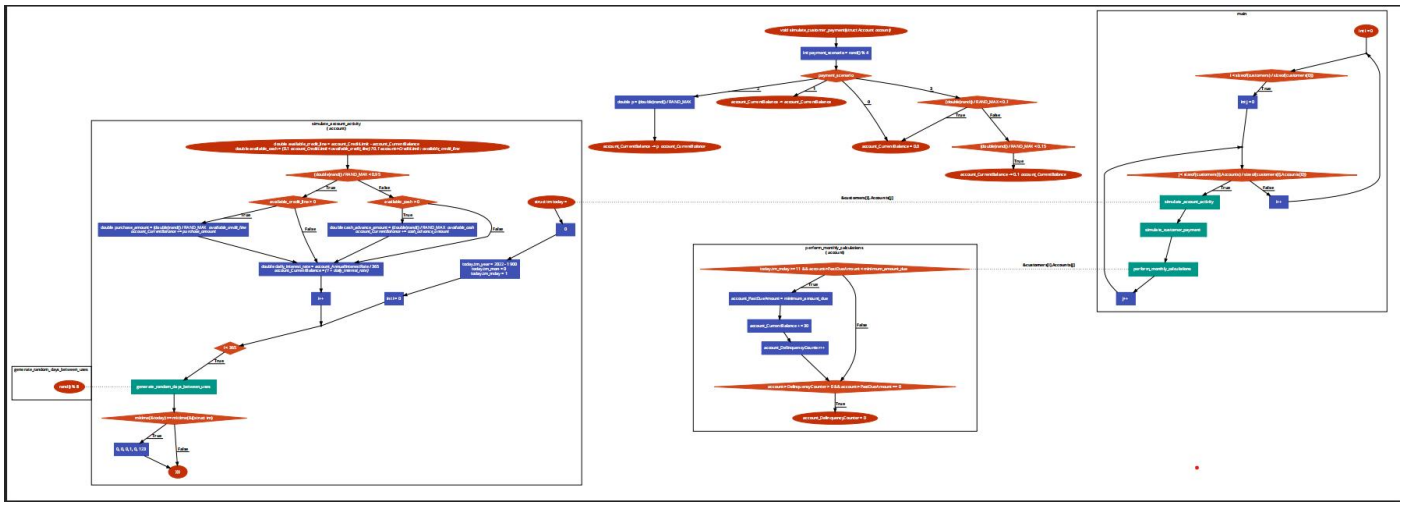
3.2 Generating Accounts

1. `**generate_random_date**(age)`: Generates a random date within a specific range based on the customer's age.
2. `**generate_account_number**(customer_id, account_index)`: Generates a unique account number based on customer ID and account index.
3. `**generate_account_credit_line**(total_credit_line)`: Generates a random account credit line within a specified range based on the total credit line.
4. `**generate_annual_fee**(account_credit_line)`: Calculates the annual fee for an account based on the account's credit line.
5. `**generate_annual_interest_rate**()`: Generates a random annual interest rate within a specified range.
6. The loop iterates over each customer and generates account information, populating the `'Accounts'` field in the customer's dictionary.
7. The inner loop generates account information for each account based on the customer's specified attributes.
8. `**account_info**`: Stores the generated account information in a dictionary.
9. `**customer['Accounts'].append(account_info)**`: Appends the generated account information to the list of accounts for the current customer.
10. The final loop generates account information for each customer, populating the `'Accounts'` field for each.



3.3 Generate Account Activity

1. `**generate_random_days_between_uses**()`: Returns a random number of days between 0 and 7.
2. `**simulate_account_activity**(account)`: Simulates a year of account activity, including purchases, cash advances, and interest calculations.
3. `**simulate_customer_payment**(account)`: Simulates various customer payment scenarios, such as exact payment, full balance payment, proportion-based payment, and payment within a specified period.
4. `**perform_monthly_calculations**(account)`: Calculates monthly account metrics including closing balance, minimum amount due, and handles delinquency scenarios.



.....

CHAPTER 4
PSEUDOCODES

4.1 Generate 20,000 customers

Define age_ranges, marital_status_probs, children_probs, education_probs

Function **generate_customer_id**():

    Return random 7-digit customer ID

Function **generate_age**():

    Return random age between 20 and 80

Function **generate_gender**():

    Return random gender ('Male' or 'Female')

Function **generate_marital_status**(age):
  For each age range, single_prob, married_prob in age_ranges, marital_status_probs:
    If age is within the range:
      Return randomly chosen marital status ('Single' or 'Married') with respective probabilities
Function **generate_children**(age):
  For each age range, probs in age_ranges, children_probs:
    If age is within the range:
      Return randomly chosen number of children based on probabilities
Function **generate_education**(age):
  For each i, (lower, upper) in age_ranges:
    If age is within the range:
      Return randomly chosen education level based on education_probs[i]
Function **generate_annual_income**(education_level, age):
  Calculate and return annual income based on education level and age
Function **generate_num_accounts**(marital_status, num_children):
  If marital_status is 'Single':
    Return num_children + 1
  Else:
    Return num_children + 2
Function **generate_total_credit_line**(num_accounts, annual_income):
  Calculate and return total credit line based on number of accounts and annual income
Initialize an empty list called customers
For each _ in range(20000):
  Generate customer data:
    - Generate customer_id
    - Generate age
    - Generate gender
    - Generate marital_status based on age
    - Generate num_children based on age
    - Generate education_level based on age
    - Generate annual_income based on education_level and age
    - Generate num_accounts based on marital_status and num_children
    - Generate total_credit_line based on num_accounts and annual_income
  Append the **generated customer data** to the **customers list**

4.2 Generating Accounts:

Define MIN_INTEREST_RATE, MAX_INTEREST_RATE
Function **generate_random_date**(age):
  Get today's date
  Calculate the earliest possible date based on age
  Generate a random date between today and the earliest date
  Return the random date
Function **generate_account_number**(customer_id, account_index):
  Concatenate customer_id and account_index and convert to an integer
  Return the generated account number
Function **generate_account_credit_line**(total_credit_line):
  Generate a random float between 0.05 and 0.5 multiplied by total_credit_line

Return the generated account credit line
Function **generate_annual_fee**(account_credit_line):
Calculate annual fee as 1% of account_credit_line
Return the generated annual fee
Function **generate_annual_interest_rate**():
Generate a random float between 0.15 and 0.3
Return the generated annual interest rate
For each customer in customers:
Initialize an empty list called 'Accounts' for the customer
For i in range(Number of Accounts for the customer):
Generate **date_opened** using **generate_random_date** with the customer's age
Generate **account_number** using **generate_account_number** with customer's ID and index i
Generate **account_credit_line** using **generate_account_credit_line** with customer's total_credit_line
Generate **annual_fee** using **generate_annual_fee** with account_credit_line
Generate **annual_interest_rate** using **generate_annual_interest_rate**
Create account_info dictionary with the following keys and corresponding values:
- 'Date Opened': date_opened
- 'Account Number': account_number
- 'Credit Line': account_credit_line
- 'Annual Fee': annual_fee
- 'Annual Interest Rate': annual_interest_rate
Append account_info to the 'Accounts' list for the customer

4.3 Generating Account Activity :

Function **generate_random_days_between_uses**():
Return random integer between 0 and 7
Function **simulate_account_activity**(account):
Set today to January 1, 2022
Repeat 365 times:
days_between_uses = generate_random_days_between_uses()
Add days_between_uses to today
If today is after January 1, 2023:
Exit loop
Calculate available_credit_line as CreditLimit minus CurrentBalance
Calculate available_cash as minimum of 10% of CreditLimit and available_credit_line
If random number between 0 and 1 is less than 0.95: # 95% chance of making a purchase
If available_credit_line > 0:
Generate random purchase_amount between 0 and available_credit_line
Add purchase_amount to CurrentBalance
Else: # 5% chance of taking cash advance
If available_cash > 0:
Generate random cash_advance_amount between 0 and available_cash
Add cash_advance_amount to CurrentBalance
Calculate daily_interest_rate as AnnualInterestRate divided by 365
Multiply CurrentBalance by (1 + daily_interest_rate)
Function **simulate_customer_payment**(account):
payment_scenario = randomly select from ['exact', 'full_balance', 'proportion', 'payment_period']

```
    If payment_scenario is 'exact':
        Set CurrentBalance to 0.0
    Else if payment_scenario is 'full_balance':
        Subtract CurrentBalance from itself
    Else if payment_scenario is 'proportion':
        Generate random proportion (p) between 0 and 1
        Multiply CurrentBalance by (1 - p)
    Else if payment_scenario is 'payment_period':
        If random number between 0 and 1 is less than 0.1:  # 10% pay entire balance within payment period
            Set CurrentBalance to 0.0
        Else if random number between 0 and 1 is less than 0.15:  # 15% pay minimum amount due within payment
period
            Subtract 10% of CurrentBalance from itself
Function perform_monthly_calculations(account):
    Calculate closing_balance as CurrentBalance
    Calculate minimum_amount_due as 10% of closing_balance
    If day of the month is greater than or equal to 11 and PastDueAmount is less than minimum_amount_due:
        Set PastDueAmount to minimum_amount_due
        Add $30 to CurrentBalance  # Late Payment Fee
        Increment DelinquencyCounter by 1
    If DelinquencyCounter is greater than 0 and PastDueAmount is 0:
        Set DelinquencyCounter to 0
For each customer in customers:
    For each account in customer's accounts:
        Set CurrentBalance to 0.0
        Set DelinquencyCounter to 0
        Set PastDueAmount to 0.0
        Call simulate_account_activity(account)
        Call simulate_customer_payment(account)
        Call perform_monthly_calculations(account)
```