Objectives & Deliverables:

In this project we will design an algorithm that can simulate (generate) a cohort of 20,000 customers along with their accounts' information and activity for 12 months.

- Generates 20,000 customers using the given characteristics.
- Generates accounts information and activity for the customers using the information given.
- Stores customers' and accounts information and activity in the right format for further analysis.

Table of Contents

1. Description of the project

The project aims to depict a real-world scenario that generates data that can be used for analytical and testing purposes. The project has been developed with the aim of producing a comprehensive report for 20,000 clients associated with Farzi Bank (FB), featuring their account details over the course of 12 months. In this project, the main aim is to deliver a proper customer dataset keeping track of various characteristics like the details of the customers, different types of accounts, credit card habits, and other key customer behaviours.

The initial segment of this project pertains to generating a client dataset. Using a realistic approach, each client is assigned a unique, 7-digit Customer ID. The age of the clients is randomly generated from a uniform distribution. The gender of the customers is determined with a 50% probability of being either male or female. The Marital Status depends upon the age of the customers, 75% of customers from the age group 20 to 30 are single and 25% are married and other age groups also have their specific marital status depending upon the probabilities. The number of children is determined using different probabilities for 20-40 and 40-80 years of age. Likewise, Education Level, Annual Income, Number of Accounts, and Total Credit Line are also determined based on the client's age, each governed by distinct probability distributions.

The second segment revolves around the Generation of Customer Accounts. To create customer accounts, several parameters are considered. The ‘Date Opened’ parameter is generated which is a random date preceding January 1, 2022, ensuring it is not older than the customer's age. The Account Number is influenced by the customer's ID. The Account Credit Line is a random proportion of the customer's Total Credit Line. The Annual Fee has a dependency on the Account Credit Line, and the Annual Interest Rate is randomly selected.

The last segment is the Generation of Account Activity. To generate the Account Activity report, a 12-month data set is collected, spanning from January 1, 2022, to December 31, 2022. Within this data set, certain key factors are considered, specifically, the number of days a customer uses their card is denoted as 'd,' which is a random integer. This variability is introduced to add a realistic approach to the simulation. In this simulation, it is assumed that when a customer uses their card, there is a high probability (0.95) that they will use it for making purchases, and a lower probability (0.05) that they will use it for cash withdrawals. There are other attributes related to this segment which are also determined based on probability.

The dataset encompasses diverse customer information, with various elements interlinked. Pseudocode and flowcharts are developed to structure and address the overall problem. The algorithm prioritizes incorporating variability and randomness to maintain data currency and produce genuine outcomes. The resulting dataset, distinguished by its uniqueness and authenticity, will serve as a valuable resource for making decisions in real-world scenarios.

2. Problem solving design

The project has been developed using Python where we have made use of Python Library for Random Data Generation. The project has been categorized the entire problem into three parts:

i. Customer Generation
ii. Account Generation
iii. Account Activity Generation

First, the information related to customers has been generated followed by their account information and how they are performing the activity in their account regarding the credit card habits like how they are using the credit card, for how many days they are using, how timely they are making their payment and similar habits.

i. Customer Generation:

The first step is the generation of customer information. In the first step, customer information is generated where we investigated the dependency of variables among each other and found out that some information related to the customers is dependent on other information like the Marital Status or Number of Children depends upon the age of the customer with different age groups having different probabilities. Likewise, there are some parameters that we have taken out without placing their dependency on any factor. All the parameters follow different probabilities.

ii. Account Generation:

After the customer information, we moved on with the customers' account information generation. This also has been generated with the help of Python where various attributes are taken into consideration. The attributes that are taken to form customer account information are the Date opened, Account number, Account credit line, Annual fee, and Annual Interest Rate. The information generated is random to give a more realistic approach. The account opening date for instance is placed before January 1st, 2022, but it shouldn't be older than the age of the customer. Likewise, we follow other attributes with unique approaches and make the customer account information.

iii. Account Activity Generation:

We have also placed our major focus on the account activity generation part of the project as it is dependent on various activities of the customers. We have investigated three factors of the account activity which are transaction, payment and monthly report. We have used a random library to generate the customers' transactions on factors such as how many days they are using their cards and for what instances they are using their cards. The account activity helps to keep track of all the purchases made by the customer in a month and all the times they have used the card to make a cash-out. In this project, we have identified and kept the track of daily transactions of the client which is derived in a random manner using Python. In addition to

that, we also have kept track of the monthly transactions for the client like closing balance, no. of purchases made, and many more.

The tracking of payments, their types, and their transactions plays a vital role in this project. Different customers have different payment types that they follow. Some customers have the habit of paying all the credit balance at the end of the same day, some pay at the end of the week, and some at the end of the month whereas there are some who do not pay at all and miss the due date. The calculations are made using Python to find out the closing balance at the end of each month. Likewise, the minimum amount due, total purchases of the month, total cash advances of the month, total advances, and similar attributes are calculated to keep proper track of the card habits of customers. There is a condition that if a customer hasn't paid the due amount, he faces delinquency. It also has been shown through this project.

We have used a realistic approach to solve the problem in this project. To make the project more like a real-world scenario, we have used unique and random data generation so that the data can give proper results. All the data generated for this project are done via Python. We first wrote down the pseudocode and made the flowchart to give the project proper result and then used Python for the proper outcome. This project is derived from the approach to solving real-world problems using Python programming language. The data derived for the project is unique and random to solve and test problems in real world.

3. Flowchart

Figure 0: Flowchart for customer and accounts details generation function

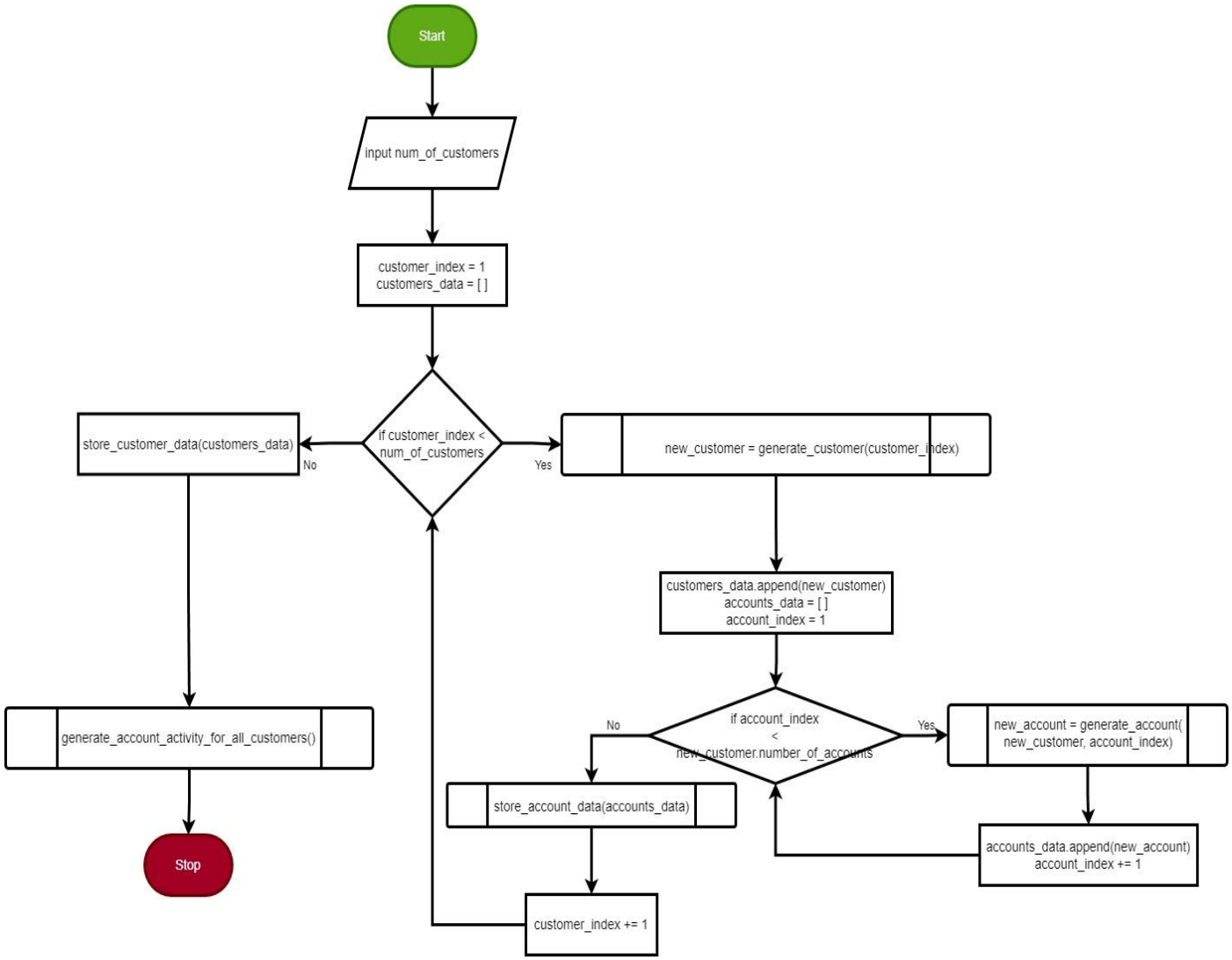


Figure 1: Flowchart for customers details generation function

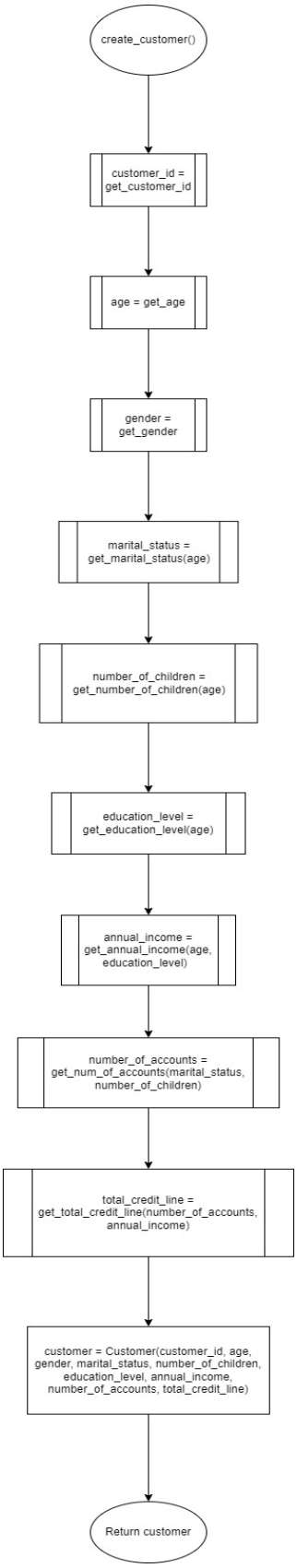


Figure 1.1: Flowchart for customer ID generation function

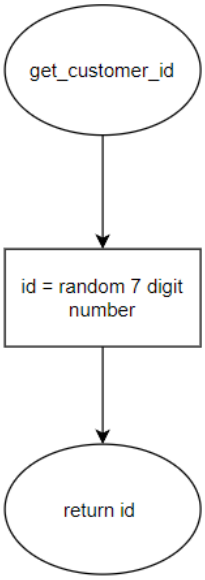


Figure 1.2: Flowchart for customer age generation function

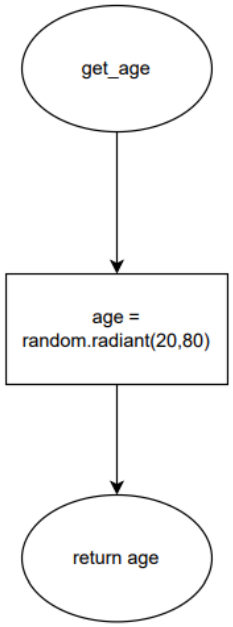


Figure 1.3: Flowchart for customer Gender generation function

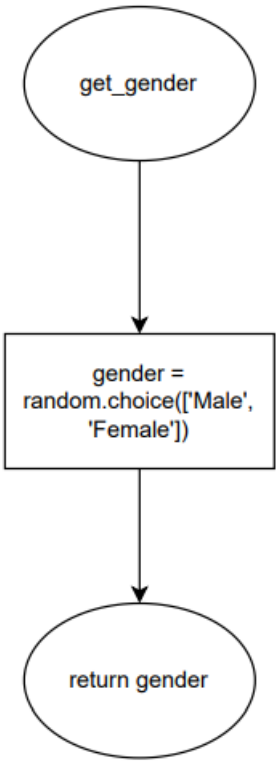


Figure 1.4: Flowchart for customer marital status generation function

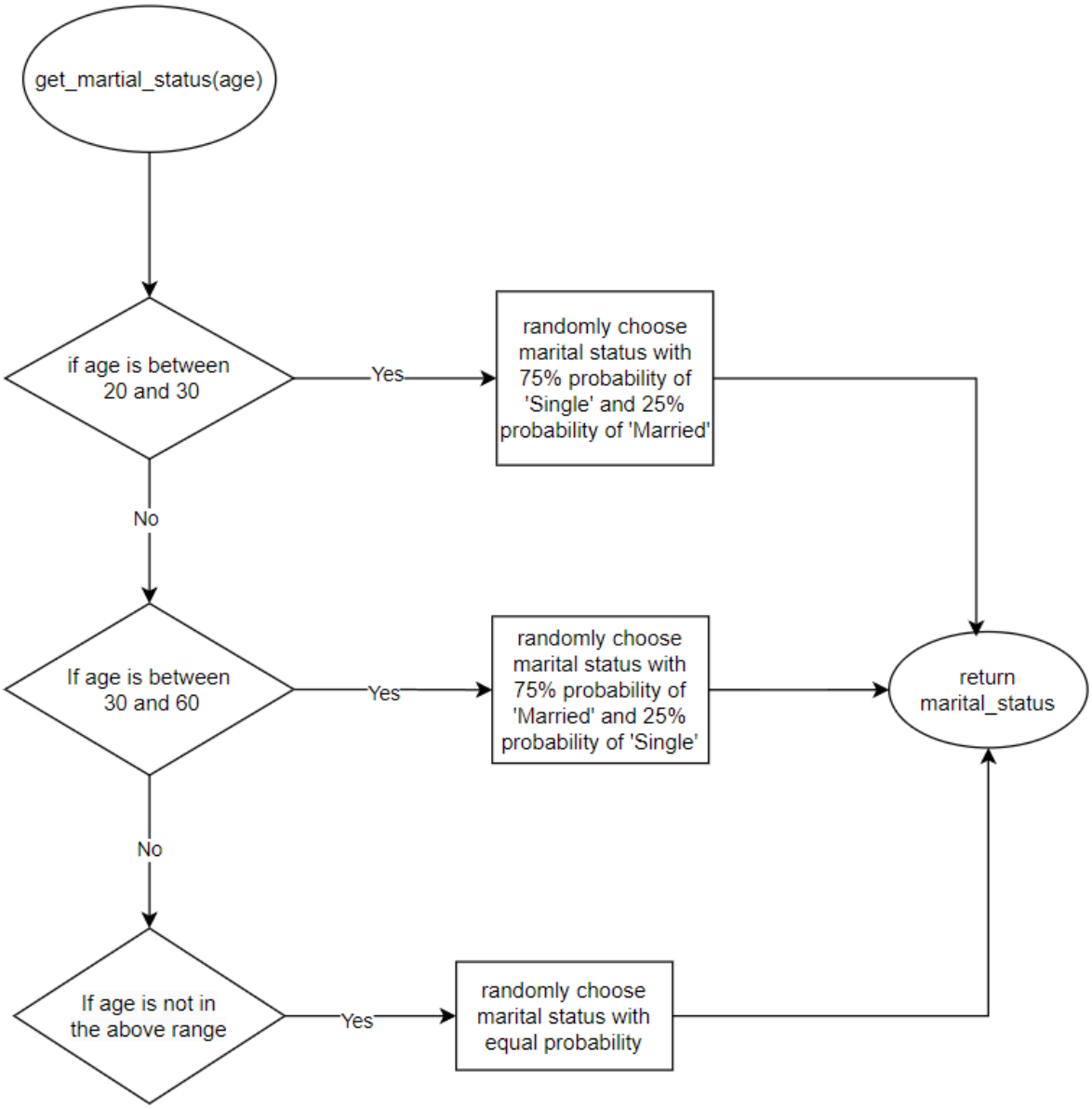


Figure 1.5: Flowchart for number of children generation function

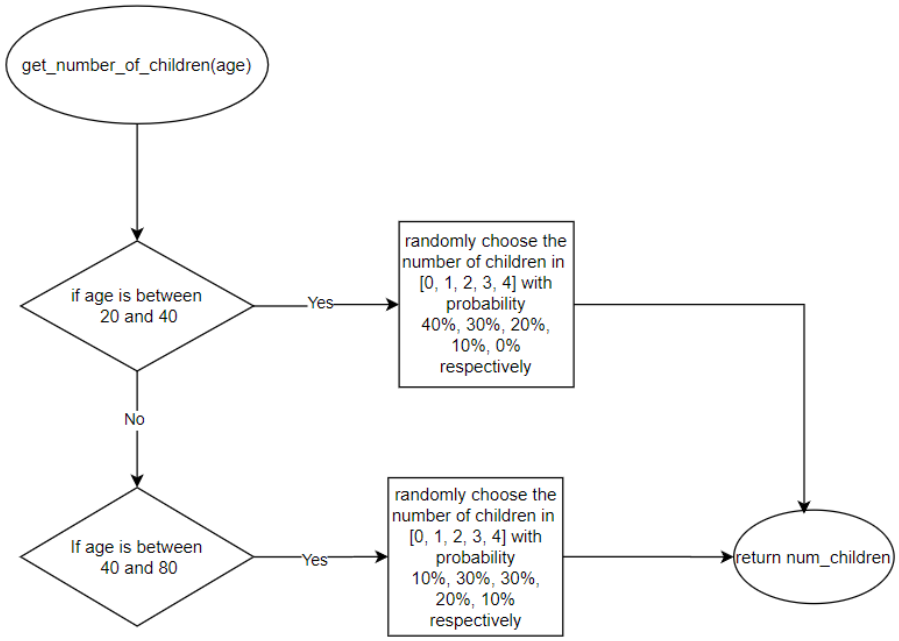


Figure 1.6: Flowchart for Education Level generation function

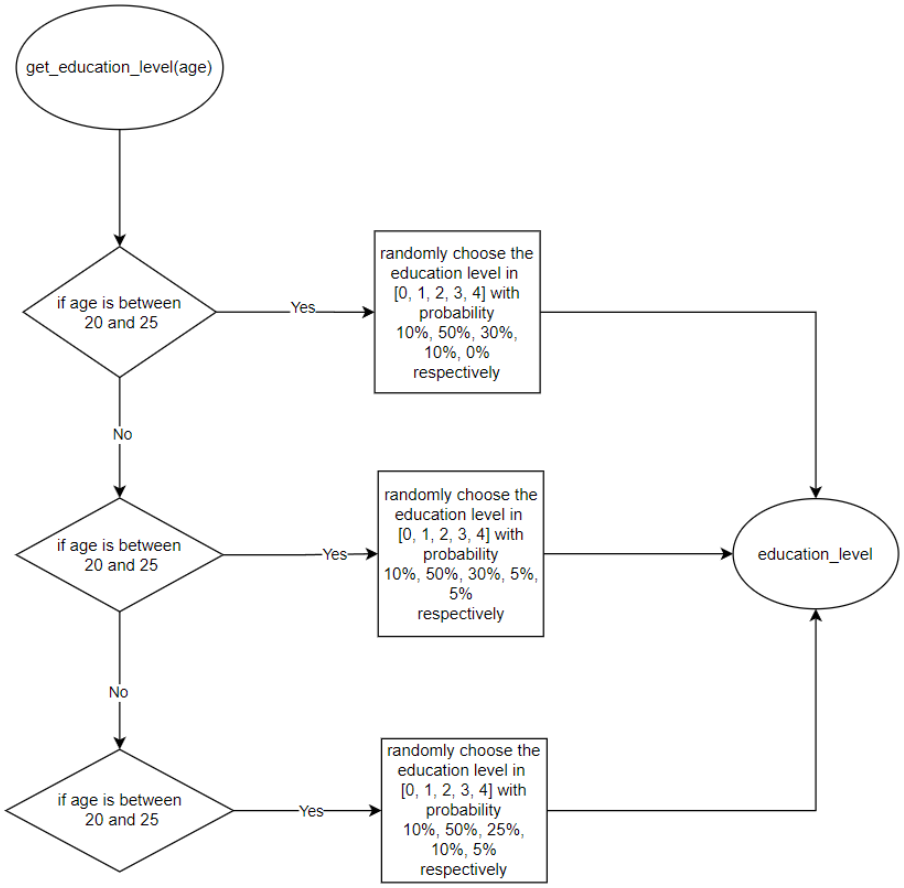


Figure 1.7: Flowchart for Annual income generation function

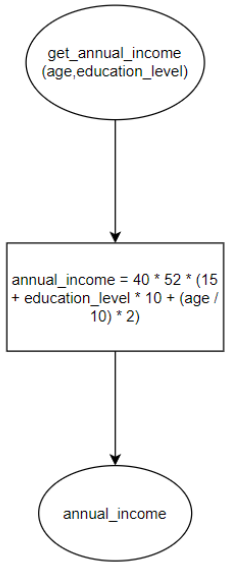


Figure 1.8: Flowchart for total credit line generation function

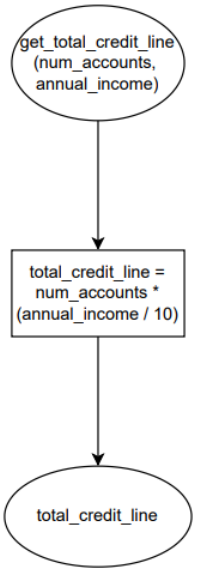


Figure 1.9: Flowchart for number of accounts generation function

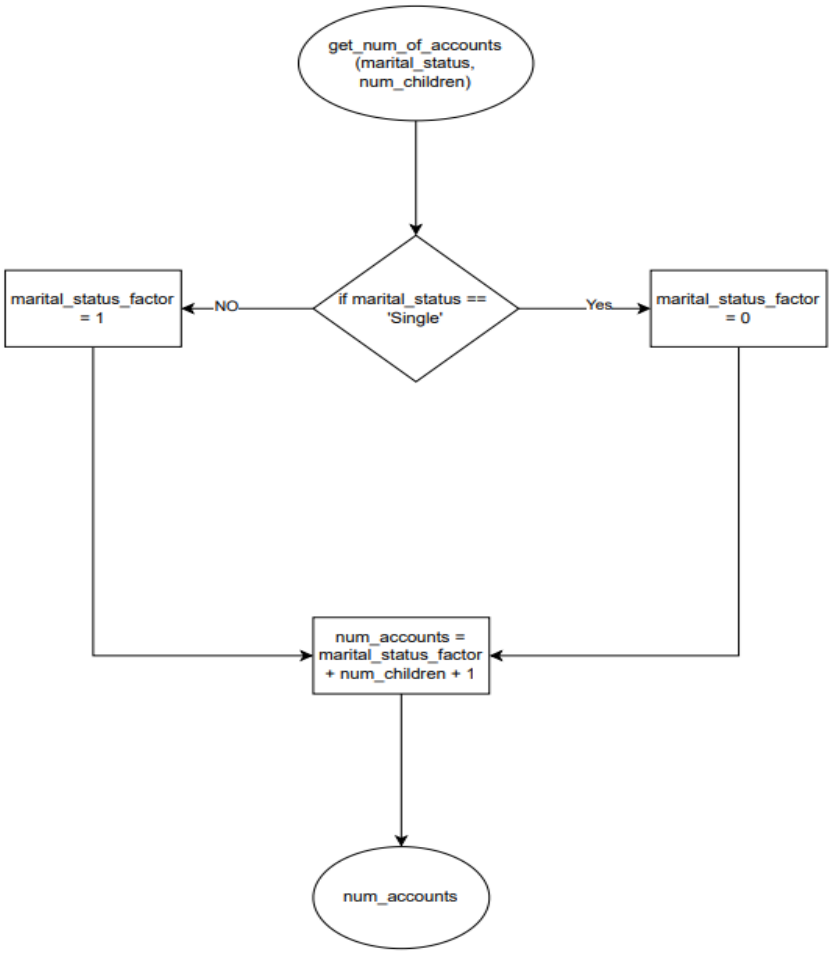


Figure 2: Flowchart for Account generation function

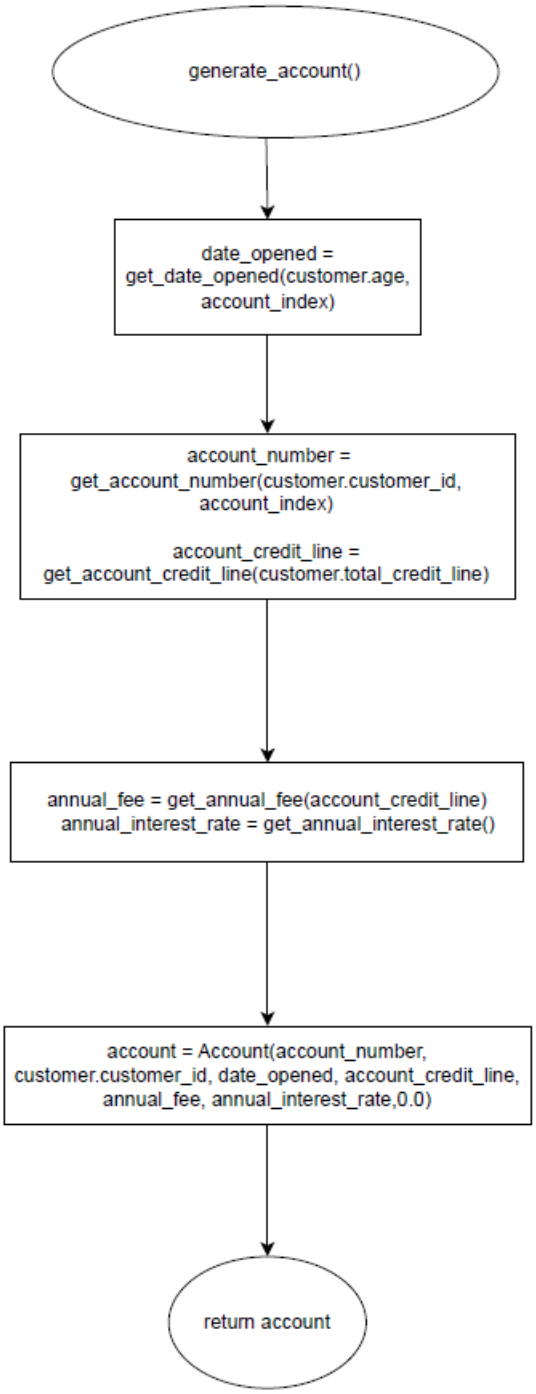


Figure 2.1: Flowchart for get_date_opened function

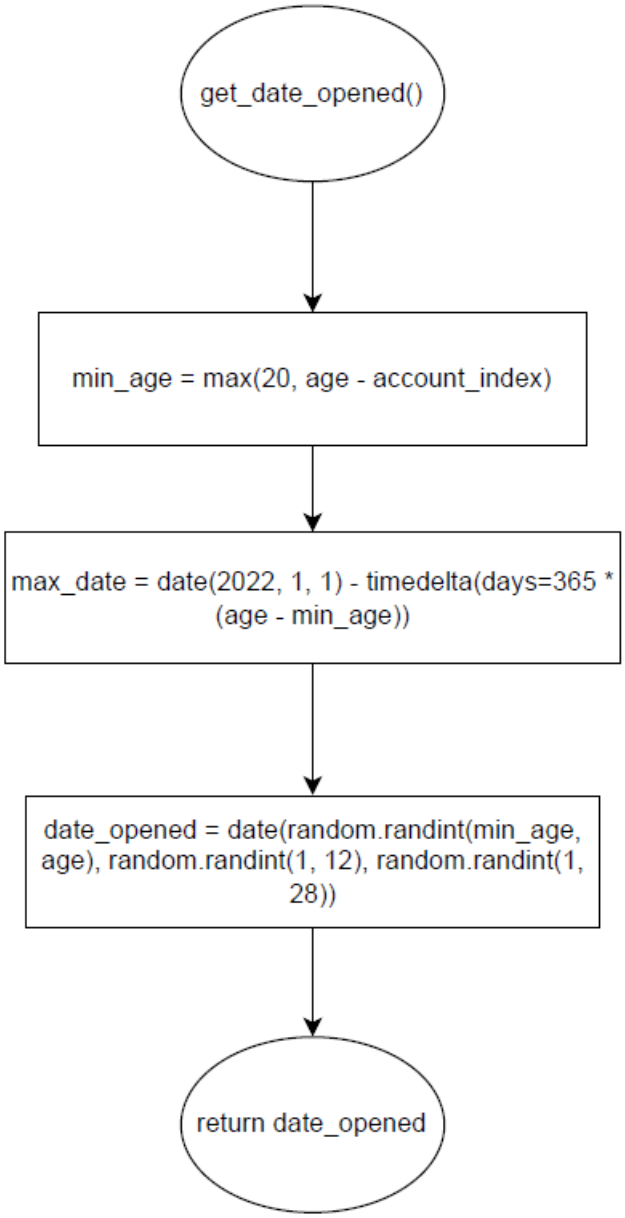


Figure 2.2: Flowchart for get_account_number function

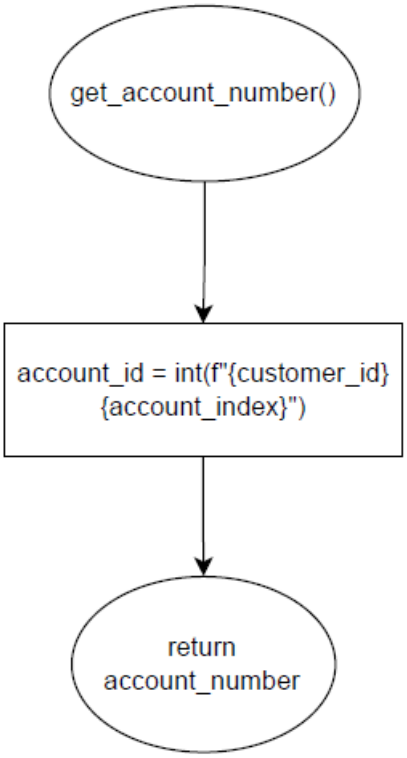


Figure 2.3: Flowchart for get_account_credit_line function

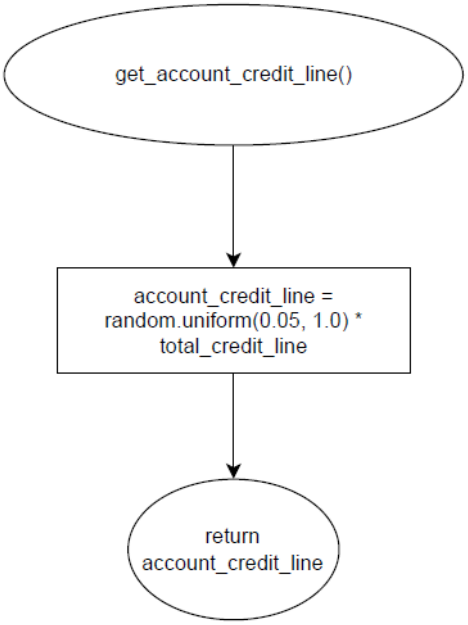


Figure 2.4: Flowchart for get_annual_fee function

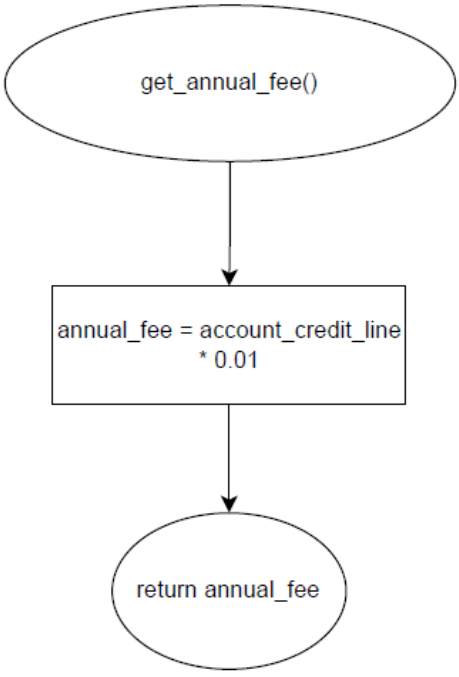


Figure 2.5: Flowchart for get_annual_iterest_rate function

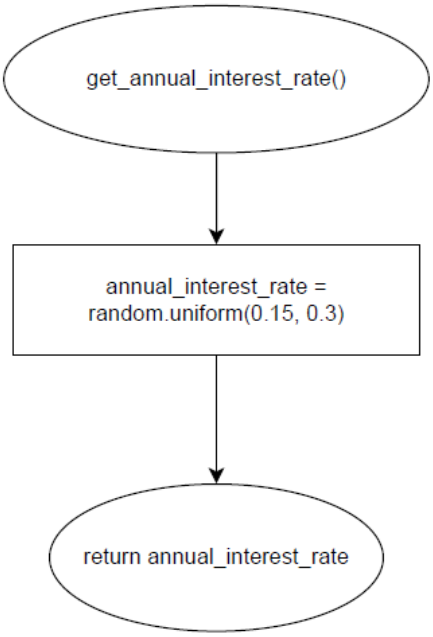


Figure 3: Flowchart for generate account activity function

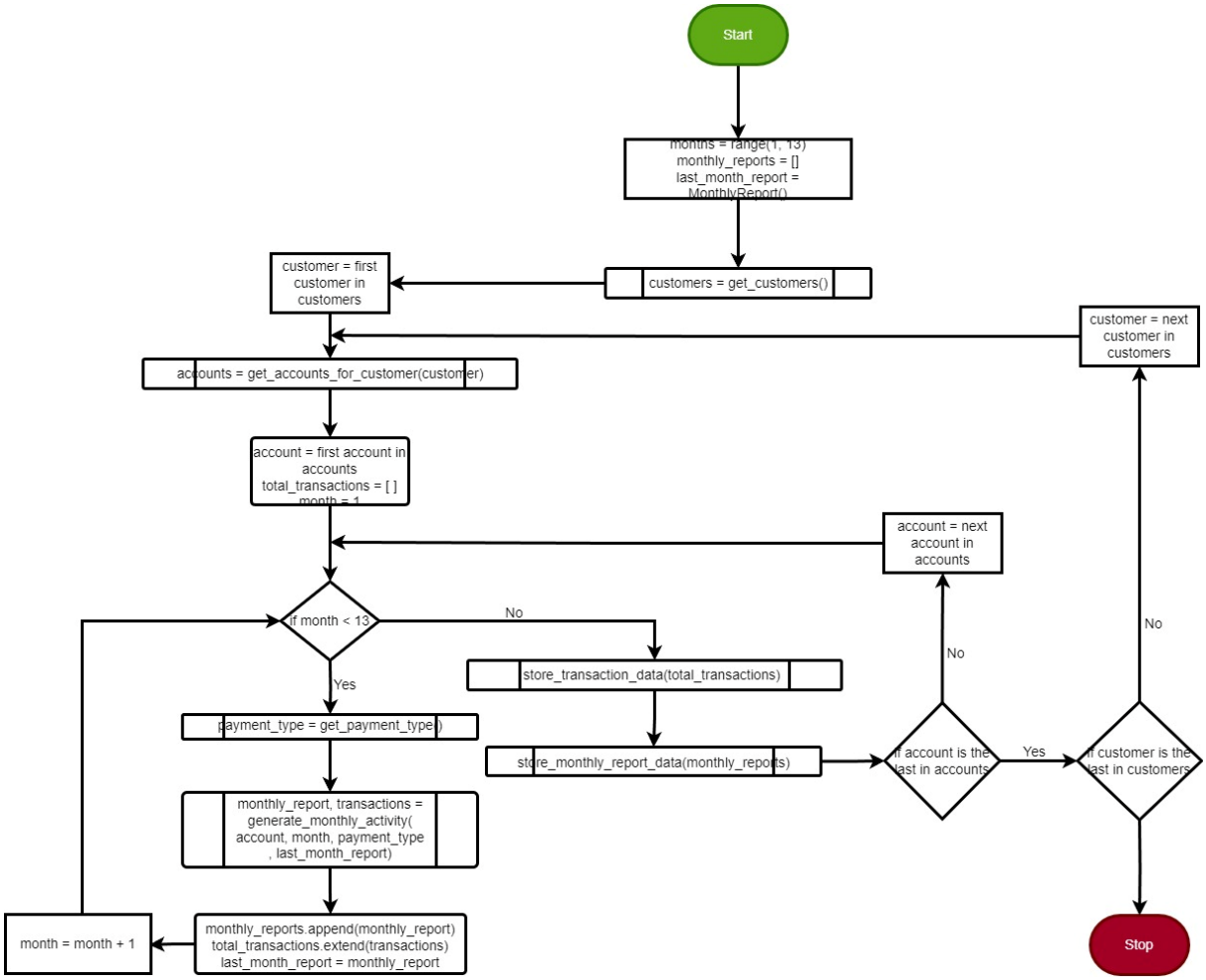


Figure 3.1: Flowchart for Customer account activity function

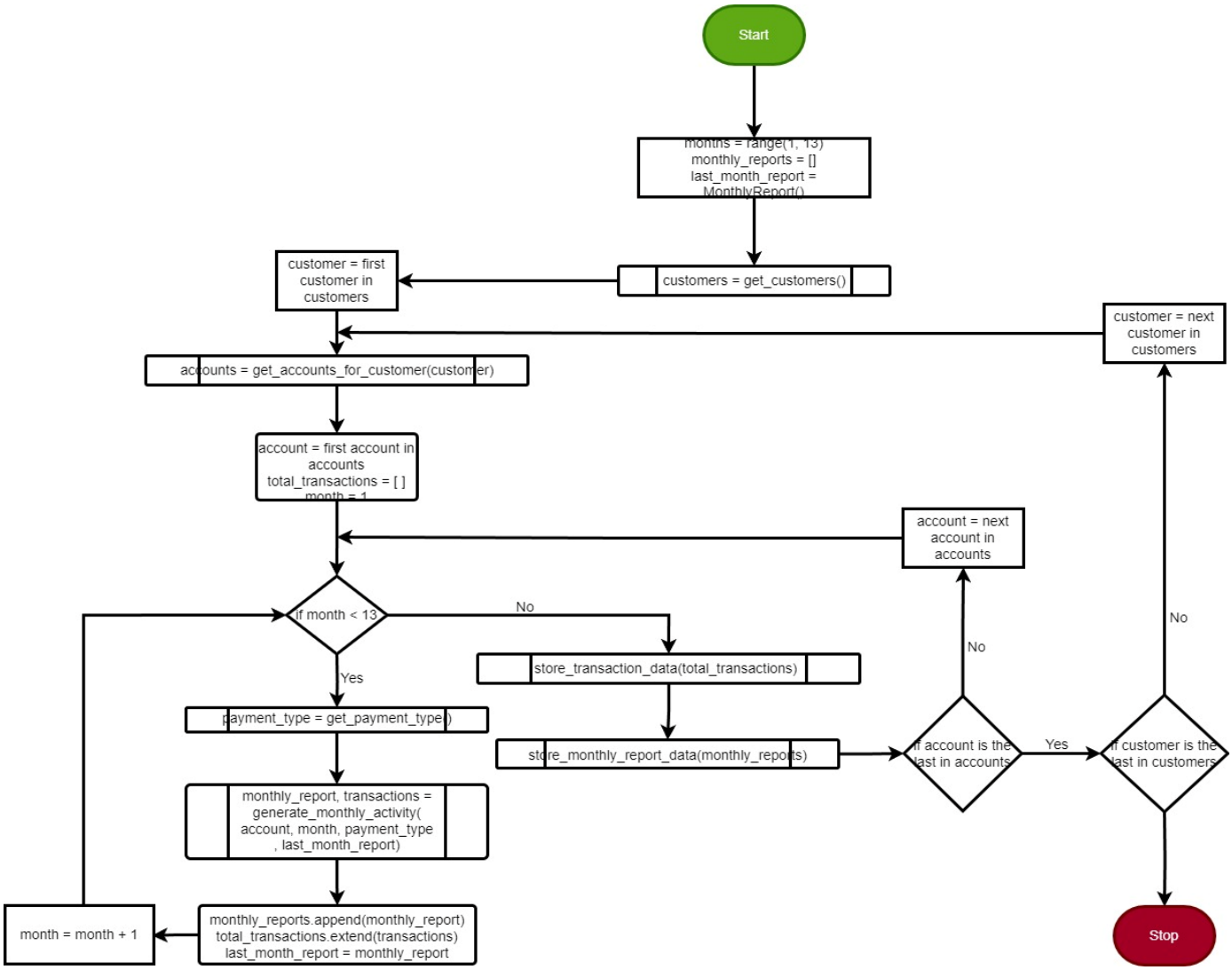


Figure 3.2: Flowchart to get monthly activity details function

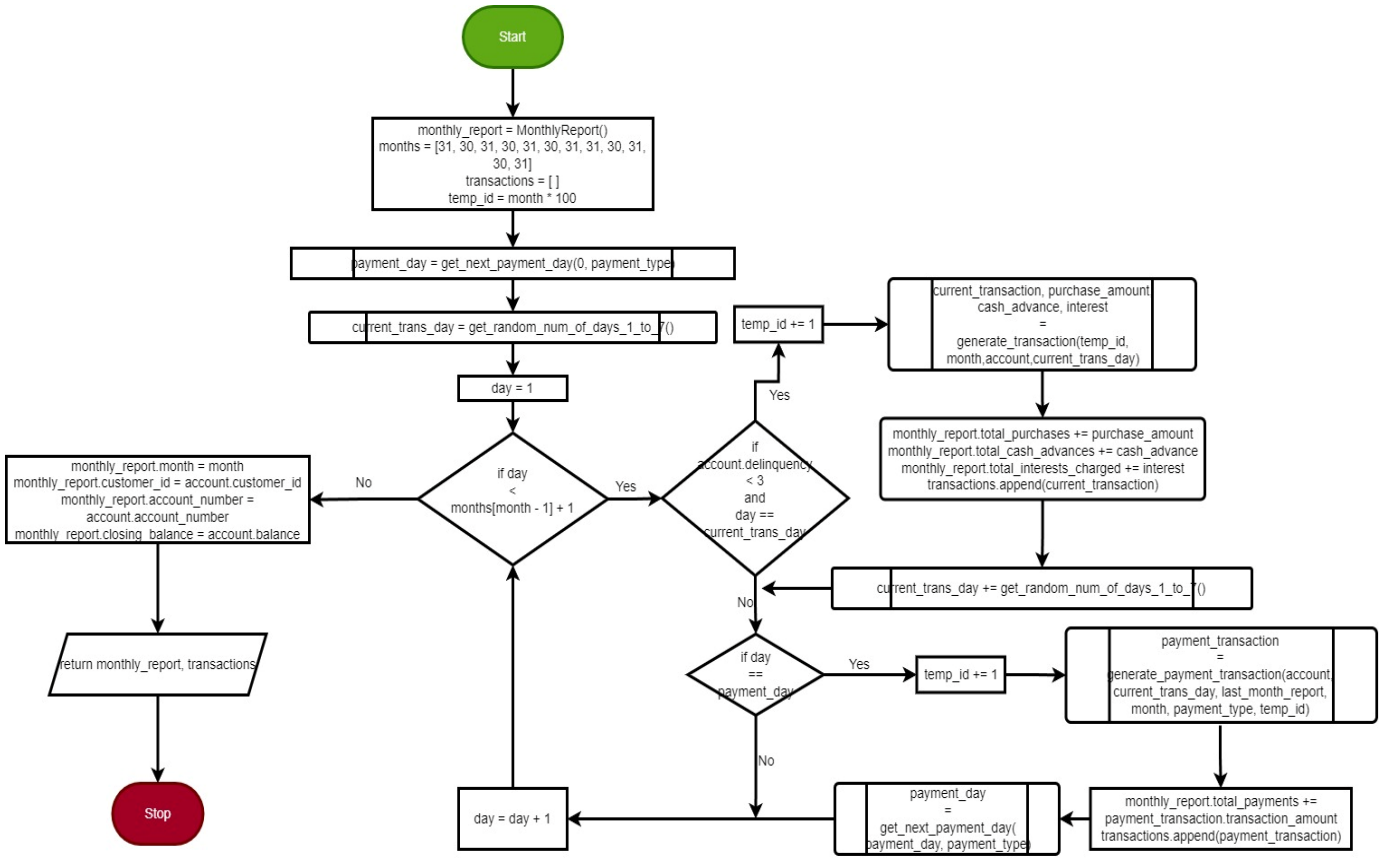


Figure 3.3: Flowchart to get transaction details function

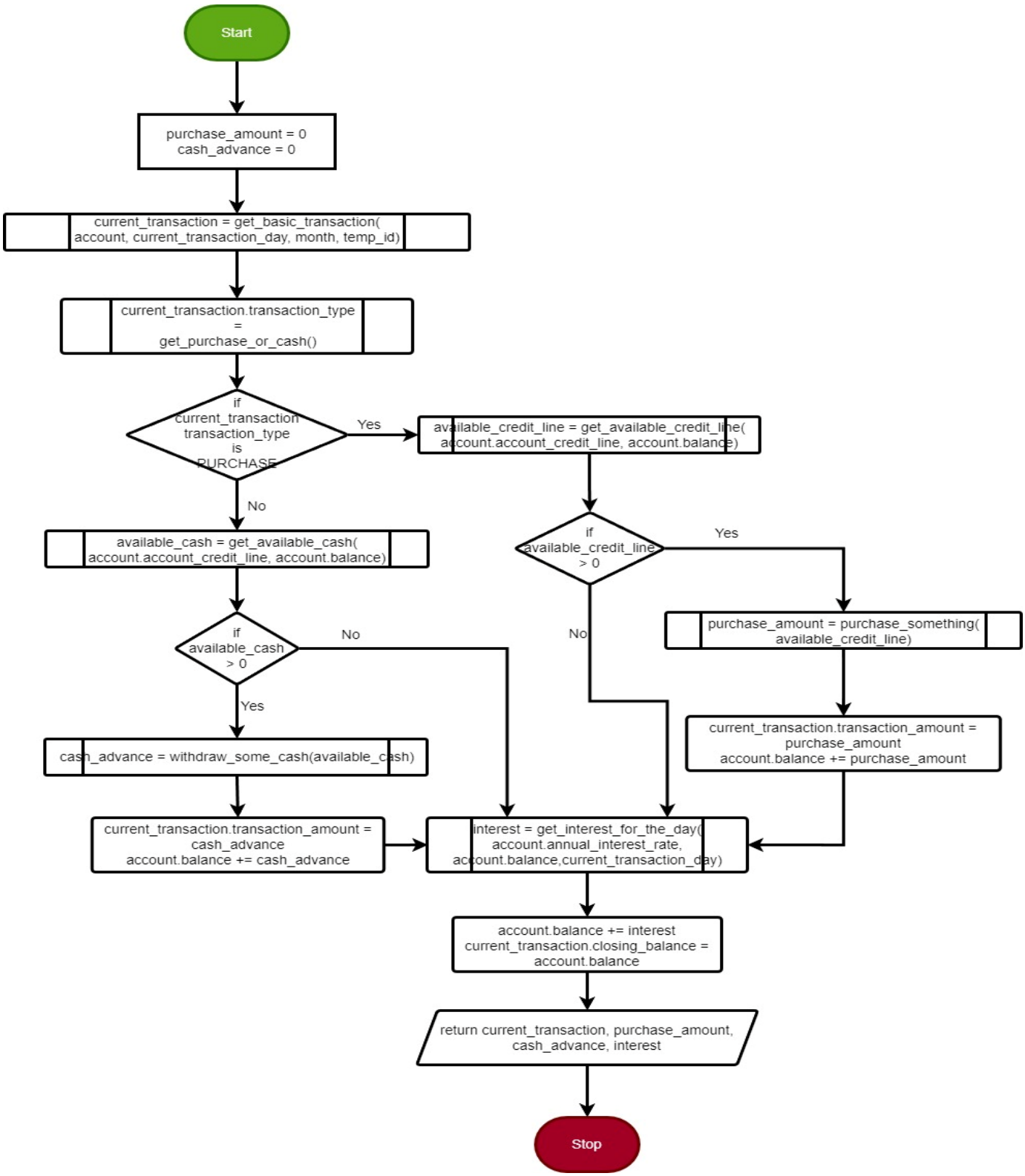


Figure 3.4: Flowchart to get payment transaction details function

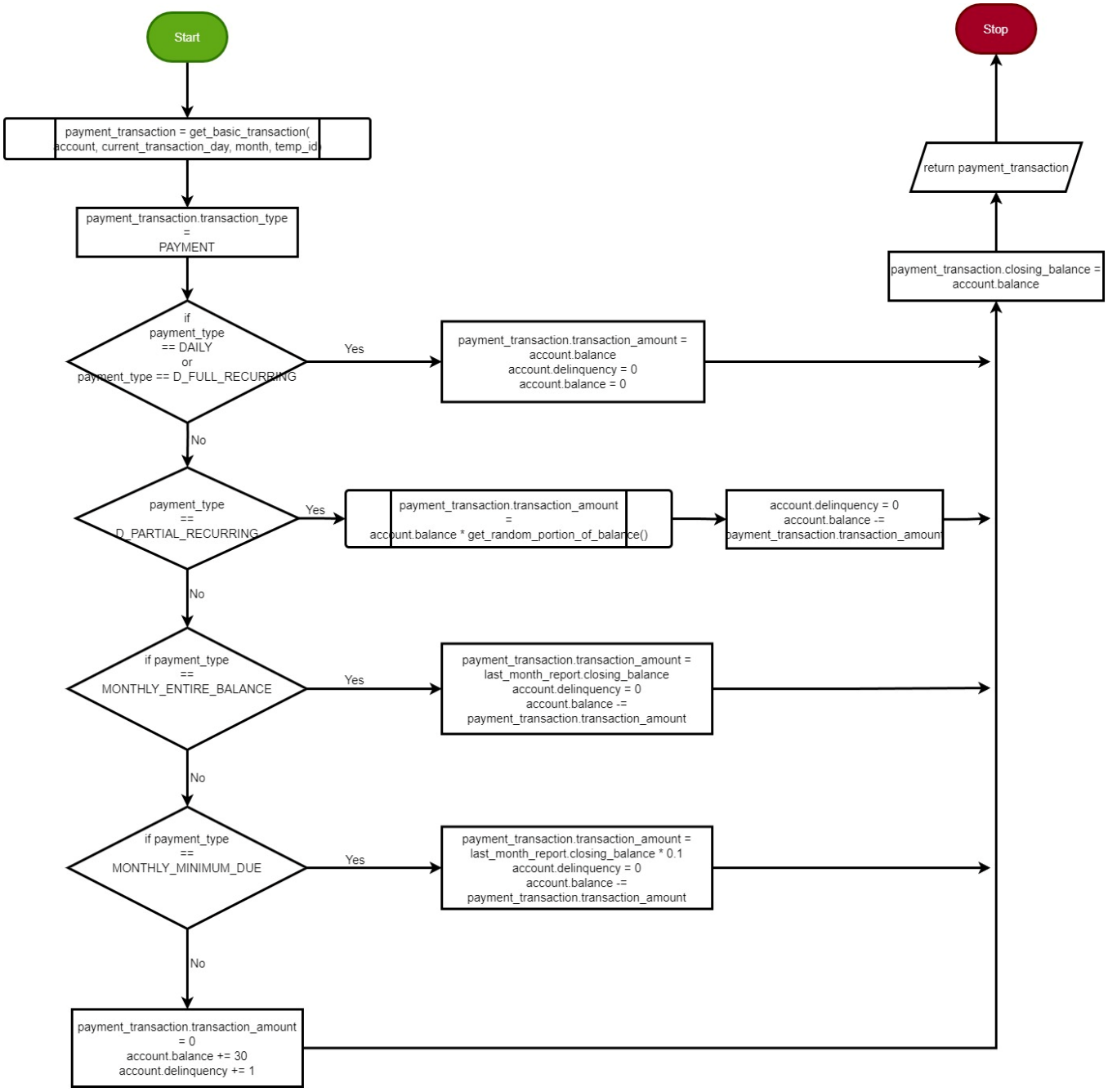


Figure 3.5: Flowchart to get basic transaction details function

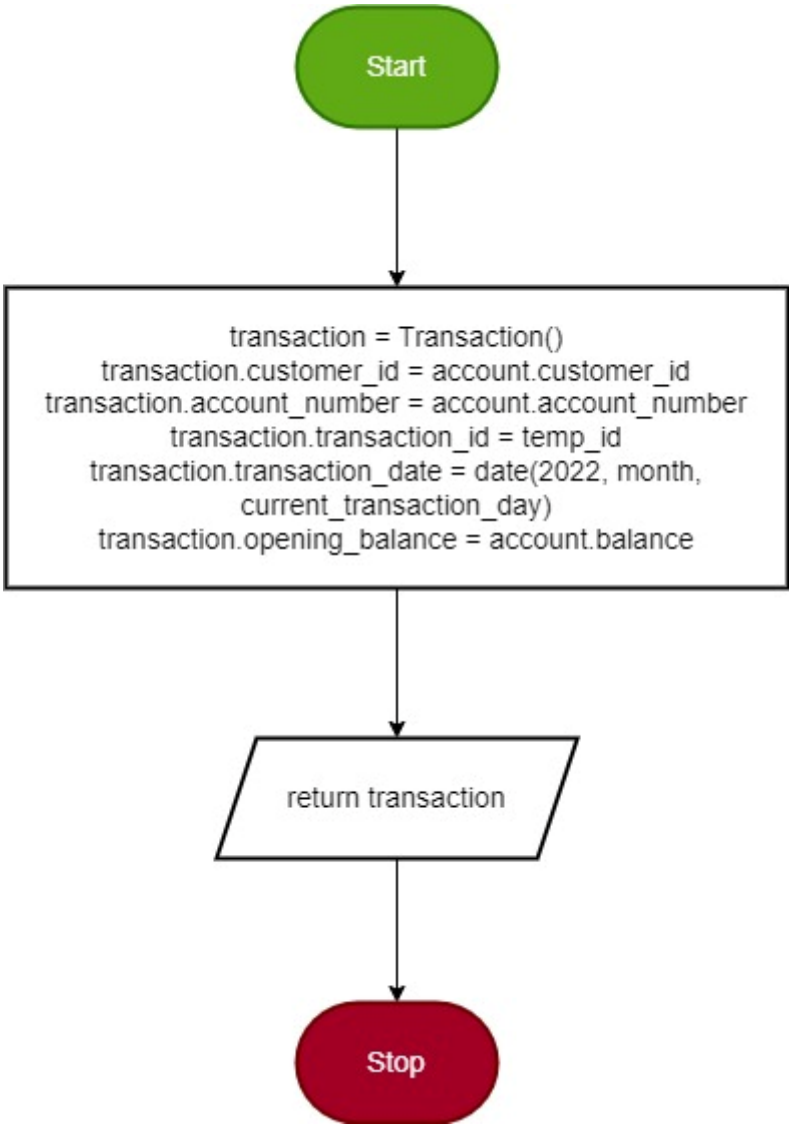


Figure 3.6: Flowchart to get next payment day details function

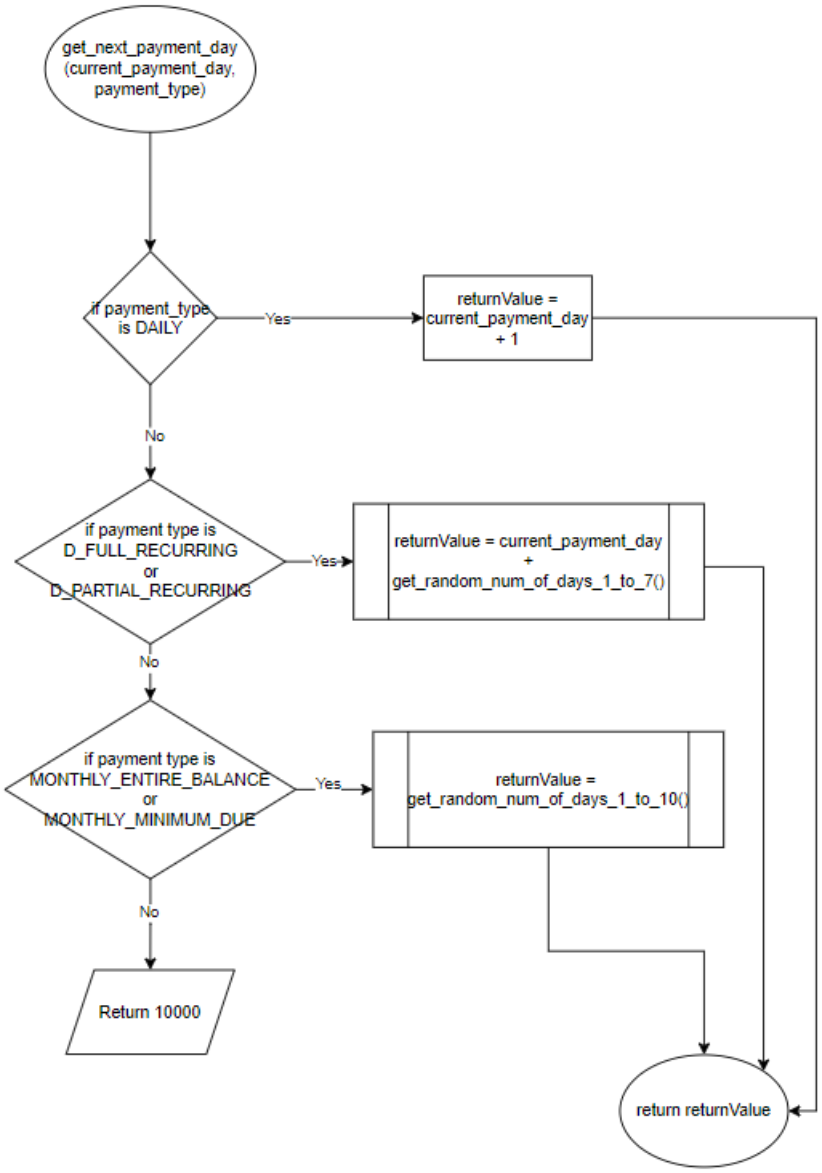


Figure 3.7: Flowchart to get payment type details function

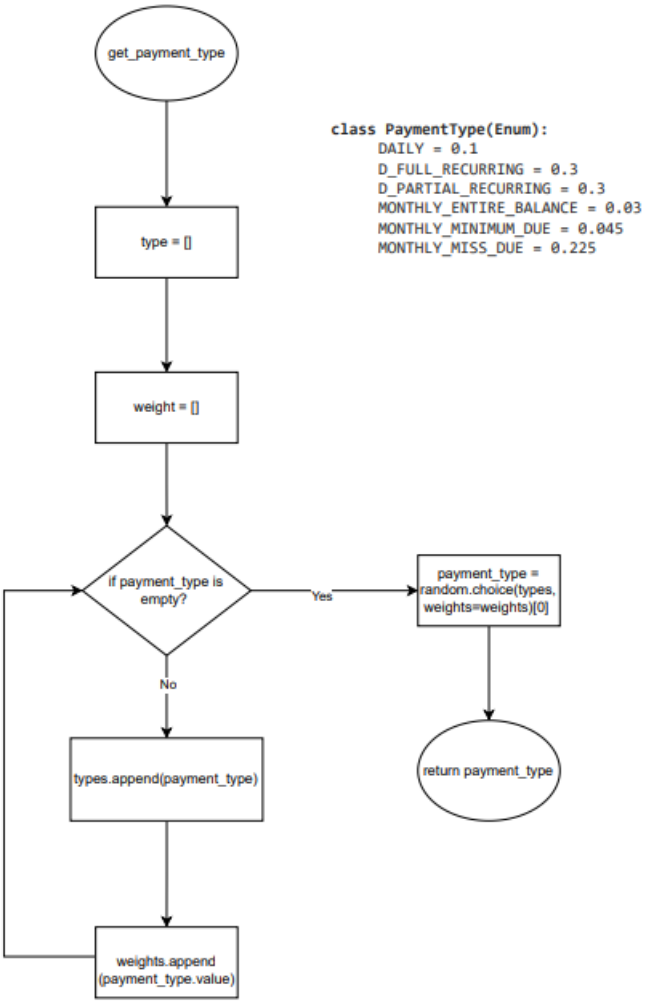


Figure 3.8: Flowchart to get random number of days details function

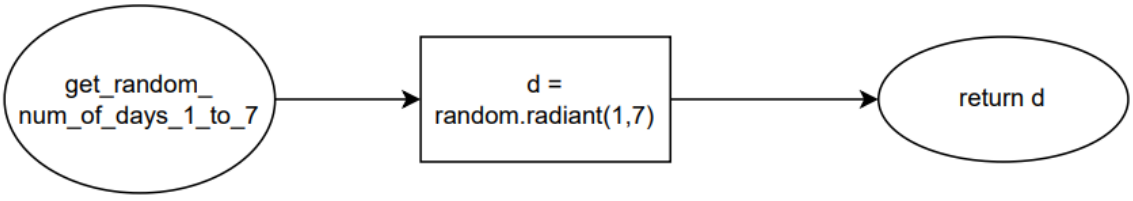


Figure 3.9: Flowchart to get random number of days details function

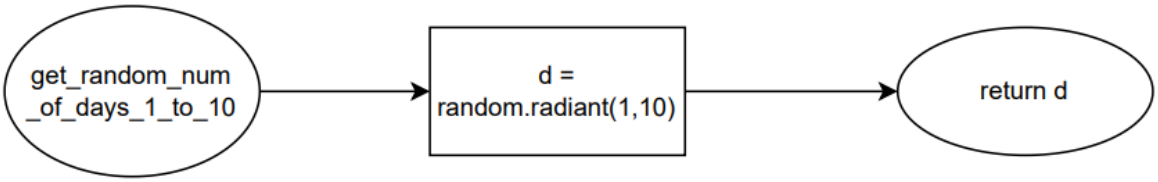


Figure 3.10: Flowchart to purchase details function

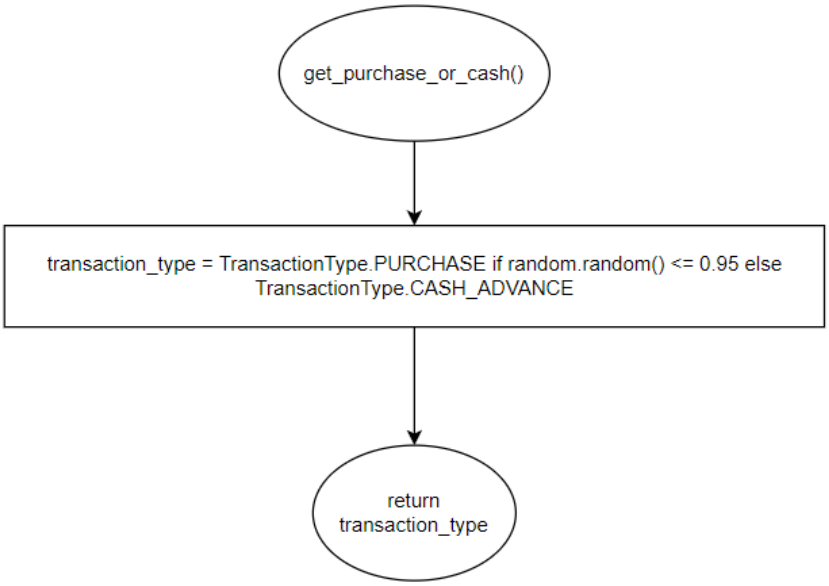


Figure 3.11: Flowchart to available credit line details function

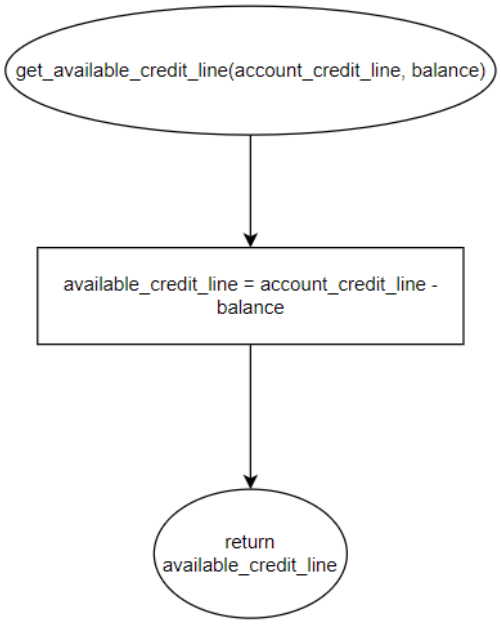


Figure 3.12: Flowchart to available Cash details function

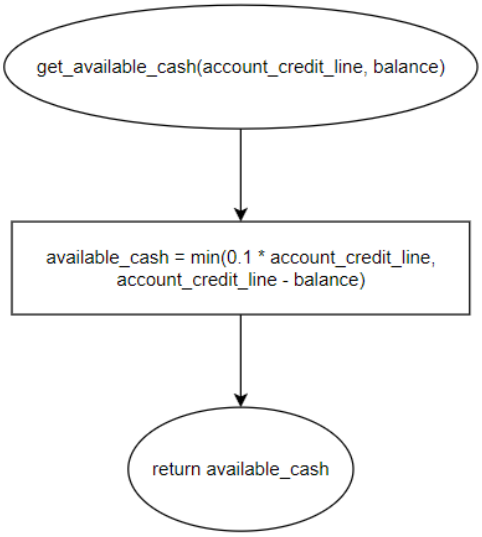
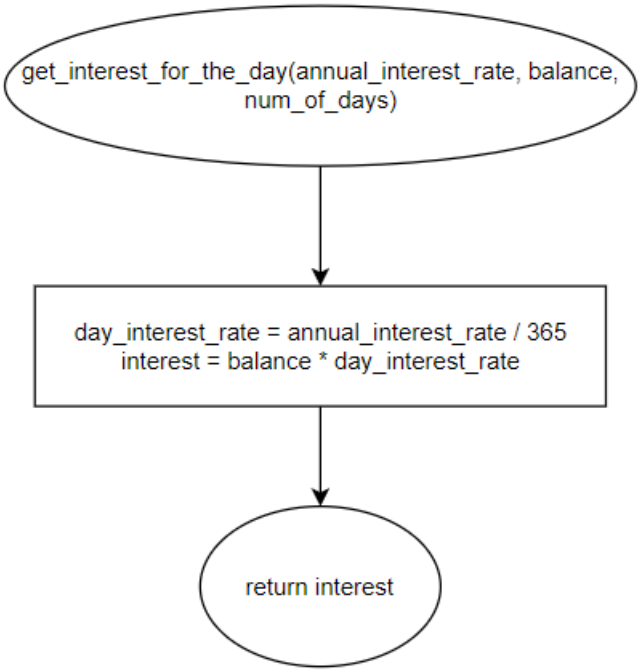


Figure 3.13: Flowchart to interest details function



4. Pseudocode

```
from ExcelUtil import *
from service.AccountGenerator import generate_account
from service.ActivityGenerator import generate_account_activity_for_all_customers
from service.CustomerGenerator import generate_customer

num_of_customers = int(input('Enter the number of customers the account data need to be generated'))
customers_data = []
for customer_index from 1 to num_of_customers:
    new_customer = generate_customer(customer_index)
    customers_data.append(new_customer)
    accounts_data = []
    for account_index from 1 to new_customer.number_of_accounts:
        new_account = generate_account(new_customer, account_index)
        accounts_data.append(new_account)
    store_account_data(accounts_data)
store_customer_data(customers_data)
generate_account_activity_for_all_customers()
```

i. Customer Generation

```
def generate_customer(customer_index):
    customer_id = get_customer_id(customer_index)
    age = get_age()
    gender = get_gender()
    marital_status = get_martial_status(age)
    number_of_children = get_number_of_children(age)
    education_level = get_education_level(age)
    annual_income = get_annual_income(age, education_level)
    number_of_accounts = get_num_of_accounts(marital_status, number_of_children)
    total_credit_line = get_total_credit_line(number_of_accounts, annual_income)
    customer = Customer(customer_id, age, gender, marital_status, number_of_children, education_level, annual_income,
                        number_of_accounts, total_credit_line)
    return customer
```

```
from Models import Customer

def get_customer_id(customer_index):
    return unique seven digit number


def get_age():
    return random number between 20 and 80 included


def get_gender():
    return random value from ['Male', 'Female']


def get_martial_status(age):
    if 20 <= age <= 30:
        return random value from ['Single', 'Married'] with probability [0.75, 0.25] respectively
    elif 30 < age <= 60:
        return random value from ['Single', 'Married'] with probability [0.25, 0.75] respectively
    else:
        return random value from ['Single', 'Married'] with probability [0.5, 0.5] respectively

def get_number_of_children(age):
    if 20 <= age <= 40:
        return random values from [0,1,2,3,4] with probability [0.4, 0.3, 0.2, 0.1, 0] respectively
    elif 40 < age <= 80:
        return random values from [0,1,2,3,4] with probability [0.1, 0.3, 0.3, 0.2, 0.1] respectively
```

```
def get_education_level(age):
    if 20 <= age <= 25:
        return random values from [0,1,2,3,4] with probability [0.1, 0.5, 0.3, 0.1, 0] respectively
    elif 25 < age <= 35:
        return random values from [0,1,2,3,4] with probability [0.1, 0.5, 0.3, 0.05, 0.05] respectively
    elif 35 < age <= 80:
        return random values from [0,1,2,3,4] with probability [0.1, 0.5, 0.25, 0.1, 0.05] respectively


def get_annual_income(age, education_level):
    return 40 * 52 * (15 + education_level * 10 + (age / 10) * 2)


def get_num_of_accounts(marital_status, num_children):
    if marital_status == 'Single':
        marital_status_factor = 0
    else:
        marital_status_factor = 1
    return marital_status_factor + num_children + 1


def get_total_credit_line(num_accounts, annual_income):
    return num_accounts * (annual_income / 10)
```

ii. Account Generation

```
def generate_account(customer, account_index):
    date_opened = get_date_opened(customer.age, account_index)
    account_number = get_account_number(customer.customer_id, account_index)
    account_credit_line = get_account_credit_line(customer.total_credit_line)
    annual_fee = get_annual_fee(account_credit_line)
    annual_interest_rate = get_annual_interest_rate()
    account = Account(account_number, customer.customer_id, date_opened,
                      account_credit_line, annual_fee, annual_interest_rate,0.0)
    return account
```

```
from Models import Account

def get_date_opened(age, account_index):
    return random date after customers 20 years age before January 1st 2022


def get_account_number(customer_id, account_index):
    return str(customer_id) + str(account_index)


def get_account_credit_line(total_credit_line):
    return random portion of total_credit_line


def get_annual_fee(account_credit_line):
    return account_credit_line * 0.01


def get_annual_interest_rate():
    return random percentage between 15 and 30 included
```

iii. Account Activity Generation

```
def generate_account_activity_for_all_customers():
    months = [1,2,3,4,5,6,7,8,9,10,11,12]
    customers = get_customers()
    monthly_reports = []
    last_month_report = MonthlyReport()
    for customer in customers:
        accounts = get_accounts_for_customer(customer)
        for account in accounts:
            total_transactions = []
            for month in months:
                payment_type = get_payment_type()
                monthly_report, transactions = generate_monthly_activity(
                                                    account, month, payment_type,
                                                    last_month_report)
                monthly_reports.append(monthly_report)
                total_transactions.extend(transactions)
                last_month_report = monthly_report
            store_transaction_data(total_transactions)
            store_monthly_report_data(monthly_reports)
```

```
def generate_monthly_activity(account, month, payment_type, last_month_report):
    monthly_report = MonthlyReport()
    current_trans_day = get_random_num_of_days_1_to_7()
    payment_day = get_next_payment_day(0, payment_type)
    total_day_in_month = [31, 30, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    transactions = []
    temp_id = month * 100
    for day from 1 to total_day_in_month:
        if account.delinquency < 3 or day == current_trans_day:
            temp_id += 1
            current_transaction, purchase_amount, cash_advance, interest = generate_transaction(temp_id, month,
                                                                            account,
                                                                            current_trans_day)
            monthly_report.total_purchases += purchase_amount
            monthly_report.total_cash_advances += cash_advance
            monthly_report.total_interests_charged += interest
            transactions.append(current_transaction)
            current_trans_day += get_random_num_of_days_1_to_7()
        if day == payment_day:
            temp_id += 1
            payment_transaction = generate_payment_transaction(account, current_trans_day, last_month_report,
                                                               month, payment_type, temp_id)
            monthly_report.total_payments += payment_transaction.transaction_amount
            transactions.append(payment_transaction)
            payment_day = get_next_payment_day(payment_day, payment_type)
    monthly_report.month = month
    monthly_report.customer_id = account.customer_id
    monthly_report.account_number = account.account_number
    monthly_report.closing_balance = account.balance
    return monthly_report, transactions
```

```python
def generate_transaction(temp_id, month, account, current_transaction_day):
    purchase_amount = 0
    cash_advance = 0
    current_transaction = get_basic_transaction(account, current_transaction_day, month, temp_id)
    current_transaction.transaction_type = get_purchase_or_cash()
    if current_transaction.transaction_type is TransactionType.PURCHASE:
        available_credit_line = get_available_credit_line(account.account_credit_line, account.balance)
        if available_credit_line > 0:
            purchase_amount = purchase_something(available_credit_line)
            current_transaction.transaction_amount = purchase_amount
            account.balance += purchase_amount
    else:
        available_cash = get_available_cash(account.account_credit_line, account.balance)
        if available_cash > 0:
            cash_advance = withdraw_some_cash(available_cash)
            current_transaction.transaction_amount = cash_advance
            account.balance += cash_advance
    interest = get_interest_for_the_day(account.annual_interest_rate, account.balance)
    account.balance += interest
    current_transaction.closing_balance = account.balance
    return current_transaction, purchase_amount, cash_advance, interest

def generate_payment_transaction(account, current_transaction_day, last_month_report, month, payment_type, temp_id):
    payment_transaction = get_basic_transaction(account, current_transaction_day, month, temp_id)
    payment_transaction.transaction_type = TransactionType.PAYMENT
    if payment_type == PaymentType.DAILY or payment_type == PaymentType.D_FULL_RECURRING:
        payment_transaction.transaction_amount = account.balance
        account.delinquency = 0
        account.balance = 0
    elif payment_type == PaymentType.D_PARTIAL_RECURRING:
        payment_transaction.transaction_amount = account.balance * get_random_portion_of_balance()
        account.delinquency = 0
        account.balance -= payment_transaction.transaction_amount
    elif payment_type == PaymentType.MONTHLY_ENTIRE_BALANCE:
        payment_transaction.transaction_amount = last_month_report.closing_balance
        account.delinquency = 0
        account.balance -= payment_transaction.transaction_amount
    elif payment_type == PaymentType.MONTHLY_MINIMUM_DUE:
        payment_transaction.transaction_amount = last_month_report.closing_balance * 0.1
        account.delinquency = 0
        account.balance -= payment_transaction.transaction_amount
    else:
        payment_transaction.transaction_amount = 0
        account.balance += 30
        account.delinquency += 1

    payment_transaction.closing_balance = account.balance
    return payment_transaction

def get_basic_transaction(account, current_transaction_day, month, temp_id):
    transaction = Transaction()
    transaction.customer_id = account.customer_id
    transaction.account_number = account.account_number
    transaction.transaction_id = temp_id
    transaction.transaction_date = date(2022, month, current_transaction_day)
    transaction.opening_balance = account.balance
    return transaction
```

```
from Enums import *
from ExcelUtil import *
from Models import MonthlyReport, Transaction


def get_random_num_of_days_1_to_7():
    return random value from 1 to 7


def get_random_num_of_days_1_to_10():
    return  random value from 1 to 10


def get_random_portion_of_balance():
    return random portion from 0 to 1


def get_purchase_or_cash():
    return random value from [TransactionType.PURCHASE, TransactionType.CASH_ADVANCE]
                             with probability [0.95,0.5] respectively


def get_available_credit_line(account_credit_line, balance):
    return account_credit_line - balance


def get_available_cash(account_credit_line, balance):
    return min(0.1 * account_credit_line, account_credit_line - balance)


def get_interest_for_the_day(annual_interest_rate, balance):
    return balance * annual_interest_rate / 365
def purchase_something(available_credit_line):
    return random value from 0 to available_credit_line


def withdraw_some_cash(available_cash):
    return random value from 0 to available_cash


def get_payment_type():
    return randon value from [DAILY,D_FULL_RECURRING,
                              D_PARTIAL_RECURRING,MONTHLY_ENTIRE_BALANCE,MONTHLY_MINIMUM_DUE,MONTHLY_MISS_DUE]
            with probability [0.1,0.3,0.3,0.03,0.045,0.225] respectively


def get_next_payment_day(current_payment_day, payment_type):
    if payment_type == PaymentType.DAILY:
        return current_payment_day + 1
    elif payment_type == PaymentType.D_FULL_RECURRING or payment_type == PaymentType.D_PARTIAL_RECURRING:
        return current_payment_day + get_random_num_of_days_1_to_7()
    elif payment_type == PaymentType.MONTHLY_ENTIRE_BALANCE or payment_type == PaymentType.MONTHLY_MINIMUM_DUE:
        return get_random_num_of_days_1_to_10()
    else:
        return INT_MAX
```

5. Reference

"A Novel Approach to Customer Segmentation in Banking" by John Smith

"Generating Synthetic Customer Data for Credit Risk Assessment" by Robert

"Python for Finance" by Yves Hilpisch

"Python for Business Analysis and Data Science" by Udemy Platform