Homework: Credit Card Payment Calculator

Table of Contents

Instructions

For this homework you will write a script to do some useful calculations for credit card payments. Use the template provided below. At a minimum, your script should contain the functions `get_min_payment()`, `interest_charged()`, `remaining_payments()`, and `main()`, each of which is described below. Be sure to include a docstring in each function.

Note: The resulting script is only an example of a remaining payments calculator and may not accurately reflect the same calculation methods as your personal credit card. This script should not be used to make financial decisions.

`get_min_payment()`

Parameters

- `balance`: The total amount of the balance in an account that is left to pay (called the *balance*; you can assume this is a positive number)
- `fees`: The fees associated with the credit card account (you can assume this is a positive integer; assign this parameter a default value of 0)

Functionality

1. Compute the minimum credit card payment. (`min_payment`). You should use the formula `min_payment = ((b * m) + f)` where
    - `b` is the balance in the account
    - `m` is the percent of the balance that needs to be paid (represented as a float where 2% is represented as .02). Make the constant value of this variable *.02*. Note: This is not the same thing as the APR.
    - `f` is the amount of fees that will be paid per month.

2. Determine if the minimum payment that was computed using the formula is below 25. If is is, we should set the minimum payment to 25 instead. Return this number.

NOTE	This function takes into account only balance, the minimum payment percentage, and fees. In the real world, minimum credit card payments may take more factors into account.

`interest_charged()`

Parameters

- `balance`: The balance of the credit card (the amount in the account that has not been paid off yet; you can assume this is a positive number)
- `apr`: The annual APR (you can assume this is an integer between 0 and 100; for example, an APR of 5% would be expressed as 5)

Functionality

- Compute and return `i`, the amount of interest accrued in the next payment according to the formula `i = (a/y)*b*d`, where
  - `a` is the APR expressed as a floating point number(for example, an APR of 8% would be .08)
  - `y` is the amount of days in a year
  - `b` is the balance in the account
  - `d` is the number of days in a billing cycle (expressed as an integer; you can safely assume that this will be 30 each time)

`remaining_payments()`

Parameters

- `balance`: The balance of the credit card (that amount in the account that has not been paid off yet; you can assume this is a positive number)
- `apr`: The annual APR (you can assume this is an integer between 0 and 100; for example, an APR of 5% would be expressed as 5)
- `targetamount`: The target payment (the amount the user wants to pay per payment; you can assume this is a positive number, though it may be passed in as a none)
- `credit_line`: The credit line. The maximum amount of balance that an account holder can keep in their account. (you can assume this is a positive integer; defaults to 5000)
- `fees`: The amount of fees that will be charged in addition to the minimum payment. (you can assume this is a positive integer; defaults to 0)

Functionality

Compute and return the number of payments required to pay off the credit card balance. We will do this by simulating payments one at a time until the balance of the credit card reaches zero. We will be assuming fixed payments (in other words, assume that each payment is the same amount of

money as the previous one) if and only if the user specified a target amount. Otherwise, the minimum payment will change according to the balance that remains in the account.

Note that (in our application) credit card payments are broken down into two parts: an interest payment, and a part that pays down the balance of the account. The interest payment is calculated as described under `interest_charged()`; the rest of the payment goes toward the balance of the credit card.

NOTE	This may not be interest is computed in the real world. For the purposes of our program we will assume that this is how it is computed.

Along with computing the number of months(payments) required to pay off the balance, this function should also compute how many months(payment periods) the balance spends over 75%, 50% and 25% of the maximum allowed credit line.

Here is an algorithm for simulating payments until the credit card is paid off:

- Initialize a counter with a value of zero; this counter represents the number of payments to be made.
- Initialize 3 counters with a value of zero; these counters represent the number of months that the balance remains over 25%, 50% and 75%, these counters are completely independent of one another and also independent of the counter that represents the number of payments to be made.
- As long as the balance of the credit card is positive, repeat the following:
    - Check if the target amount parameter was passed in as *None*, if it was passed in as *None*, use the get_min_payment() function to get the min payment based on the current balance and the fees. The payment amount will either be the minimum payment, or the target amount if it was passed in.
    - Use the interest_charged() function to determine what portion of the next payment will be interest. The total payment minus interest charged is the amount that will go toward paying the balance. Remember, these amounts will change with every payment.
    - If the payment towards the balance is negative, then this means that the balance increased even after payment. Given the parameter values, the balance will never be paid off. If this is the case, print a message to the user stating that the card balance cannot be paid off and quit the script. Otherwise, the program may continue.
    - Reduce the balance by the part of the payment that goes toward paying the balance.
    - Check if the balance is greater than 75% of the credit line, if so, increase the appropriate counter.
    - Check if the balance is greater than 50% of the credit line, if so, increase the appropriate counter.
    - Check if the balance is greater than 25% of the credit line, if so, increase the appropriate counter.
    - Increase the counter that counts the number of payments that have been performed.
- When the balance of the is no longer positive, the value of the counter is the number of

payments required. Return the counters all together as a tuple.

NOTE	You can return multiple items at once from a function if you separate each item by a comma. The item returned will be a tuple containing the items that you returned. Each element in the tuple can be accessed referencing its respective index.

`main()`

Parameters

- `balance`: The balance of the credit card (that amount in the account that has not been paid off yet; you can assume this is a positive number)
- `apr`: The annual APR (you can assume this is an integer between 0 and 100; for example, an APR of 5% would be expressed as 5)
- `targetamount`: The target payment (the amount the user wants to pay per payment; you can assume this is a positive number; defaults to None)
- `credit_line`: The credit line. The maximum amount of balance that an account holder can keep in their account. (you can assume this is a positive integer; defaults to 5,000)
- `fees`: The amount of fees that will be charged in addition to the minimum payment. (you can assume this is a positive integer; defaults to 0)

Functionality

- Compute the recommended minimum payment using the `get_min_payment()` function
- Display the recommended minimum payment to the user
- Declare a variable named `pays_minimum` to False. This variable represents whether a user has chosen to pay the minimum payment each month or not.
- If the user's target payment is `None`, set the `pays_minimum` to True. Otherwise, if the user's target payment is less than the minimum payment, print a message to the user (e.g., "Your target payment is less than the minimum payment for this credit card") and quit the program; otherwise:
  - Use `remaining_payments()` to figure out the total number of payments required (hint: we will assume that the beginning balance is the balance amount that was passed in).
    - If the user pays the minimum payment each time, display the number that represents the number of payments that need to be made to the user (e.g., "If you pay the minimum payments each month, you will pay off the balance in < *total payments* > payments."; replace the angle brackets with the appropriate values)
    - If the user does not pay the minimum payments, display the number that represents the number of payments that need to be made to the user along with what the desired payment amount is (e.g., "If you make payments of $< *target payment* >, you will pay off the balance in < *total payments* > payments."; replace the angle brackets with the appropriate values)
  - Return a string that is a message that will tell the user how many payments they will be above 25, 50 and 75 percent thresholds.

NOTE	Your returned string should mimic the structure and content of the output provided in the example output below.

Template

```
"""Perform credit card calculations."""
from argparse import ArgumentParser
import sys

# replace this comment with your implementation of get_min_payment(),
# interest_charged(), remaining_payments(), and main()
def parse_args(args_list):
    """Takes a list of strings from the command prompt and passes them through as
    arguments

    Args:
        args_list (list) : the list of strings from the command prompt
    Returns:
        args (ArgumentParser)
    """
    parser = ArgumentParser()

    parser.add_argument('balance_amount', type = float, help = 'The total amount of
balance left on the credit account')
    parser.add_argument('apr', type = int, help = 'The annual APR, should be an int
between 1 and 100')
    parser.add_argument('credit_line', type = int, help = 'The maximum amount of
balance allowed on the credit line.')
    parser.add_argument('--payment', type = int, default = None, help = 'The amount
the user wants to pay per payment, should be a positive number')
    parser.add_argument('--fees', type = float, default = 0, help = 'The fees that are
applied monthly.')

    # parse and validate arguments
    args = parser.parse_args(args_list)

    if args.balance_amount < 0:
        raise ValueError("balance amount must be positive")
    if not 0 <= args.apr <= 100:
        raise ValueError("APR must be between 0 and 100")
    if args.credit_line < 1:
        raise ValueError("credit line must be positive")
    if args.payment is not None and args.payment < 0:
        raise ValueError("number of payments per year must be positive")
    if args.fees < 0:
        raise ValueError("fees must be positive")

    return args
```

```
if __name__ == "__main__":

    try:
        arguments = parse_args(sys.argv[1:])
    except ValueError as e:
        sys.exit(str(e))

    print(main(arguments.balance_amount, arguments.apr, credit_line = arguments
.credit_line, targetamount = arguments.payment, fees = arguments.fees))
```

Running your program

Your program is designed to run from the terminal. To run it, open a terminal and ensure you are in the directory where your script is saved.

The program takes three required command-line arguments: a balance amount, an APR integer (expressed as a number between 0 and 100) and a credit line amount (expressed as a positive integer). It also allows the following optional arguments: `--payment` (the desired monthly payment) and `--fees` (the amount of monthly fees that the credit card requires). Below are some examples of how to use the program. The examples assume you are using macOS and your program is called `credit_card.py`. If you are using Windows, replace `python3` with `python`. If your program has a different name, replace `credit_card.py` with the name of your program.

Basic usage

```
python3 credit_card.py 15_000 10 17_000
```

Output:

```
Your recommended starting minimum payment is $300.0.
If you pay the minimum payments each month, you will pay off the credit card in 275
payments

You will spend a total of 106 months over 25% of the credit line
You will spend a total of 47 months over 50% of the credit line
You will spend a total of 13 months over 75% of the credit line
```

With one optional parameter

```
python3 credit_card.py 15_000 10 17_000 --payment 300
```

Output:

```
Your recommended starting minimum payment is $300.0.
If you make payments of $300, you will pay off the credit card in 65 payments

You will spend a total of 49 months over 25% of the credit line
You will spend a total of 32 months over 50% of the credit line
You will spend a total of 12 months over 75% of the credit line
```

With multiple optional parameters

```
python3 credit_card.py 15_000 10 17_000 --payment 340 --fees 20
```

Output:

```
Your recommended starting minimum payment is $320.0.
If you make payments of $340, you will pay off the credit card in 56 payments

You will spend a total of 41 months over 25% of the credit line
You will spend a total of 26 months over 50% of the credit line
You will spend a total of 10 months over 75% of the credit line
```