SPOCs - MIT 19/10/2024

Part (a)

Paying Debt off in a Year

0.0/10.0 points (graded)

Write a program to calculate the credit card balance after one year if a person only pays the minimum monthly payment required by the credit card company each month.

The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal
3. `monthlyPaymentRate` - minimum monthly payment rate as a decimal

For each month, calculate statements on the monthly payment and remaining balance. At the end of 12 months, print out the remaining balance. Be sure to print out no more than two decimal digits of accuracy - so print

```
Remaining balance: 813.41
```

instead of

```
Remaining balance: 813.4141998135
```

So your program only prints out one thing: the remaining balance at the end of the year in the format:

```
Remaining balance: 4784.0
```

A summary of the required math is found below:

Monthly interest rate= (Annual interest rate) / 12.0
Minimum monthly payment = (Minimum monthly payment rate) x (Previous balance)
Monthly unpaid balance = (Previous balance) - (Minimum monthly payment)
Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

We provide sample test cases below. We suggest you develop your code on your own machine, and make sure your code passes the sample test cases, before you paste it into the box below.

Test Cases to Test Your Code With. Be sure to test these on your own machine - and that you get the same output! - before running your code on this webpage!

Problem 1 Test Cases

Note: Depending on where you round in this problem, your answers may be off by a few cents in either direction. Do not worry if your solution is within +/- 0.05 of the correct answer. Be sure to test these on your own machine - and that you get the same output! - before running your code on this webpage!

Test Cases:

```
            # Test Case 1:
            balance = 42
            annualInterestRate = 0.2
            monthlyPaymentRate = 0.04

            # Result Your Code Should Generate Below:
            Remaining balance: 31.38

          # To make sure you are doing calculation correctly, this
is the
          # remaining balance you should be getting at each month
for this example
             Month 1 Remaining balance: 40.99
             Month 2 Remaining balance: 40.01
             Month 3 Remaining balance: 39.05
             Month 4 Remaining balance: 38.11
             Month 5 Remaining balance: 37.2
             Month 6 Remaining balance: 36.3
             Month 7 Remaining balance: 35.43
             Month 8 Remaining balance: 34.58
             Month 9 Remaining balance: 33.75
             Month 10 Remaining balance: 32.94
             Month 11 Remaining balance: 32.15
             Month 12 Remaining balance: 31.38
```

1.

```
            Test Case 2:
            balance = 484
            annualInterestRate = 0.2
            monthlyPaymentRate = 0.04

            Result Your Code Should Generate Below:
            Remaining balance: 361.61
```

2.

Code Editor

```
monthlyInterestRate = annualInterestRate / 12.0

for _ in range(12):
    minimumPayment = monthlyPaymentRate * balance
    unpaidBalance = balance - minimumPayment
    balance = unpaidBalance + (monthlyInterestRate * unpaidBalance)

print("Remaining balance:", round(balance, 2))
```

Press ESC then TAB or click outside of the code editor to exit

Hints

Only two decimal digits of accuracy?

Use the `round` function at the end of your code!

How to think about this problem?

To help you get started, here is a rough outline of the stages you should probably follow in writing your code:

- For each month:
    - Compute the monthly payment, based on the previous month's balance.
    - Update the outstanding balance by removing the payment, then charging interest on the result.
    - Output the month, the minimum monthly payment and the remaining balance.
    - Keep track of the total amount of paid over all the past months so far.
- Print out the result statement with the total amount paid and the remaining balance.

Use these ideas to guide the creation of your code.

Important

Do not define your own values

** For problems such as these, do not include `input` statements or define variables we told you would be given. Our automated testing will provide values for you - so write your code in the following box assuming those variables are already defined. The code you paste into the following box **should not** specify the values for the variables `balance`, `annualInterestRate`, or `monthlyPaymentRate`

Part (b)

Paying Debt Off in a Year

0.0/15.0 points (graded)

Now write a program that calculates the minimum **fixed** monthly payment needed in order pay off a credit card balance within 12 months. By a fixed monthly payment, we mean a single number which does not change each month, but instead is a constant amount that will be paid each month.

In this problem, we will *not* be dealing with a minimum monthly payment rate.

The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal

The program should print out one line: the lowest monthly payment that will pay off all debt in under 1 year, for example:

```
Lowest Payment: 180
```

Assume that the interest is compounded monthly according to the balance at the end of the month (after the payment for that month is made). The monthly payment must be a multiple of $10 and is the same for all months. Notice that it is possible for the balance to become negative using this payment scheme, which is okay. A summary of the required math is found below:

> **Monthly interest rate** = (Annual interest rate) / 12.0
> **Monthly unpaid balance** = (Previous balance) - (Minimum fixed monthly payment)
> **Updated balance each month** = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

Test Cases to Test Your Code With. Be sure to test these on your own machine - and that you get the same output! - before running your code on this webpage!

Problem 2 Test Cases

Be sure to test these on your own machine - and that you get the same output! - before running your code on this webpage!

Test Cases:

```
Test Case 1:
balance = 3329
annualInterestRate = 0.2

Result Your Code Should Generate:
-------------------
Lowest Payment: 310
```

1.

```
Test Case 2:
balance = 4773
annualInterestRate = 0.2

Result Your Code Should Generate:
-------------------
Lowest Payment: 440
```

2.

```
Test Case 3:
balance = 3926
annualInterestRate = 0.2

Result Your Code Should Generate:
-------------------
Lowest Payment: 360
```

3.

Code Editor

```
balance = 3329
annualInterestRate = 0.2

monthlyInterestRate = annualInterestRate / 12.0
fixedPayment = 10

while True:
    testBalance = balance
```

```
    for _ in range(12):
        testBalance = (testBalance - fixedPayment) * (1 +
monthlyInterestRate)
    if testBalance <= 0:
        break
    fixedPayment += 10

print("Lowest Payment:", fixedPayment)
```


Hints

Hint: How to think about this problem?

- Start with $10 payments per month and calculate whether the balance will be paid off in a year this way (be sure to take into account the interest accrued each month).
- If $10 monthly payments are insufficient to pay off the debt within a year, increase the monthly payment by $10 and repeat.

Hint: A way of structuring your code

- If you are struggling with how to structure your code, think about the following:
    - Given an initial balance, what code would compute the balance at the end of the year?
    - Now imagine that we try our initial balance with a monthly payment of $10. If there is a balance remaining at the end of the year, how could we write code that would reset the balance to the initial balance, increase the payment by $10, and try again (using the same code!) to compute the balance at the end of the year, to see if this new payment value is large enough.
    - **I'm still confused!**
      A good way to implement this problem will be to use a loop structure. You may want to refresh your understanding of **while** loops. Think hard about how the program will know when it has found a good minimum

monthly payment value - when a good value is found, the loop can terminate.

- Be careful - you don't want to overwrite the original value of `balance`. You'll need to save that value somehow for later reference!

Reminder: Only hit "Submit" once per submission. We are unable to give you more than 30 checks.

Part C:

Using Bisection Search to Make the Program Faster

0.0/20.0 points (graded)

You'll notice that in Problem 2, your monthly payment had to be a multiple of $10. Why did we make it that way? You can try running your code locally so that the payment can be any dollar and cent amount (in other words, the monthly payment is a multiple of $0.01). Does your code still work? It should, but you may notice that your code runs more slowly, especially in cases with very large balances and interest rates. (Note: when your code is running on our servers, there are limits on the amount of computing time each submission is allowed, so your observations from running this experiment on the grading system might be limited to an error message complaining about too much time taken.)

Well then, how can we calculate a more accurate fixed monthly payment than we did in Problem 2 without running into the problem of slow code? We can make this program run faster using a technique introduced in lecture - bisection search!

The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal

To recap the problem: we are searching for the smallest monthly payment such that we can pay off the entire balance within a year. What is a reasonable **lower bound** for this payment value? $0 is the obvious answer, but you can do better than that. If there was no interest, the debt can be paid off by monthly payments of one-twelfth of the original balance, so we must pay at least this much every month. One-twelfth of the original balance is a good lower bound.

What is a good **upper bound**? Imagine that instead of paying monthly, we paid off the entire balance at the end of the year. What we ultimately pay must be greater than what we would've paid in monthly installments, because the interest was compounded on the balance we didn't pay off each month. So a good upper bound for the monthly payment would be one-twelfth of the balance, *after* having its interest compounded monthly for an entire year.

In short:

> **Monthly interest rate** = (Annual interest rate) / 12.0
> **Monthly payment lower bound** = Balance / 12
> **Monthly payment upper bound** = (Balance x (1 + Monthly interest rate)12) / 12.0

Write a program that uses these bounds and bisection search (for more info check out the Wikipedia page on bisection search) to find the smallest monthly payment *to the cent* (no more multiples of $10) such that we can pay off the debt within a year. Try it out with large inputs, and notice how fast it is (try the same large inputs in your solution to Problem 2 to compare!). Produce the same return value as you did in Problem 2.

Note that if you do not use bisection search, your code will not run - your code only has 30 seconds to run on our servers.

Test Cases to Test Your Code With. Be sure to test these on your own machine - and that you get the same output! - before running your code on this webpage!

Problem 3 Test Cases

Note: The automated tests are lenient—if your answers are off by a few cents in either direction, your code is OK.

Be sure to test these on your machine - and that you get the same output! - before running your code on this webpage!

Test Cases:

```
Test Case 1:
balance = 320000
annualInterestRate = 0.2

Result Your Code Should Generate:
-------------------
Lowest Payment: 29157.09
```

1.

```
Test Case 2:
balance = 999999
annualInterestRate = 0.18

Result Your Code Should Generate:
-------------------
Lowest Payment: 90325.03
```

2.

By: MITx