Introduction

Each month, a credit card statement will come with the option for you to pay a minimum amount of your charge, usually 2% of the balance due. However, the credit card company earns money by charging interest on the balance that you don't pay. So even if you pay credit card payments on time, interest is still accruing on the outstanding balance.

Say you've made a $5,000 purchase on a credit card with an 18% annual interest rate and a 2% minimum monthly payment rate. If you only pay the minimum monthly amount for a year, how much is the remaining balance?

You can think about this in the following way.

At the beginning of month 0 (when the credit card statement arrives), assume you owe an amount we will call b_0 (*b* for *balance*; subscript *0* to indicate this is the balance at month 0).

Any payment you make during that month is deducted from the balance. Let's call the payment you make in month 0, p_0. Thus, your **unpaid balance** for month 0, ub_0, is equal to $b_0 - p_0$.

$$ub_0 = b_0 - p_0$$

At the beginning of month 1, the credit card company will charge you interest on your unpaid balance. So if your annual interest rate is r, then at the beginning of month 1, your new balance is your previous unpaid balance ub_0, **plus** the interest on this unpaid balance for the month. In algebra, this new balance would be

$$b_1 = ub_0 + r/12.0 \cdot ub_0$$

In month 1, we will make another payment, p_1. That payment has to cover some of the interest costs, so it does not completely go towards paying off the original charge. The balance at the beginning of month 2, b_2, can be calculated by first calculating the unpaid balance after paying p_1, then by adding the interest accrued:

$$ub_1 = b_1 - p_1$$

$$b_2 = ub_1 + r/12.0 \cdot ub_1$$

If you choose just to pay off the minimum monthly payment each month, you will see that the compound interest will dramatically reduce your ability to lower your debt.

Let's look at an example. If you've got a $5,000 balance on a credit card with 18% annual interest rate, and the minimum monthly payment is 2% of the current balance, we would have the following repayment schedule if you only pay the minimum payment each month:

Month	Balance	Minimum Payment	Unpaid Balance	Interest
0	5000.00	100 (= 5000 * 0.02)	4900 (= 5000 - 100)	73.50 (= 0.18/12.0 * 4900)
1	4973.50 (= 4900 + 73.50)	99.47 (= 4973.50 * 0.02)	4874.03 (= 4973.50 - 99.47)	73.11 (= 0.18/12.0 * 4874.03)
2	4947.14 (= 4874.03 + 73.11)	98.94 (= 4947.14 * 0.02)	4848.20 (= 4947.14 - 98.94)	72.72 (= 0.18/12.0 * 4848.20)

You can see that a lot of your payment is going to cover interest, and if you work this through month 12, you will see that after a year, you will have paid $1165.63 and yet you will still owe $4691.11 on what was originally a $5000.00 debt. Pretty depressing!

Problem 1 - Paying Debt off in a Year

10.0/10.0 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 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


               Test Case 2:
```

```
balance = 484
annualInterestRate = 0.2
monthlyPaymentRate = 0.04

Result Your Code Should Generate Below:
Remaining balance: 361.61


```

Problem 2 - Paying Debt Off in a Year

15.0/15.0 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)

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:

```
1.
2.          Test Case 1:
3.          balance = 3329
4.          annualInterestRate = 0.2
5.
6.          Result Your Code Should Generate:
7.          -------------------
8.          Lowest Payment: 310
9.
10.
11.
12.                 Test Case 2:
13.                 balance = 4773
14.                 annualInterestRate = 0.2
15.
16.                 Result Your Code Should Generate:
17.                 -------------------
18.                 Lowest Payment: 440
19.
20.
21.
22.                 Test Case 3:
23.                 balance = 3926
24.                 annualInterestRate = 0.2
25.
26.                 Result Your Code Should Generate:
27.                 -------------------
28.                 Lowest Payment: 360
29.
30.
```

Problem 3 - Using Bisection Search to Make the Program Faster

20.0/20.0 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