



---

## PROBLEM 3: USING BISECTION SEARCH TO MAKE THE PROGRAM FASTER

(25/25 points)

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)<sup>12</sup>) / 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!

[Click to See 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 own 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
```

```

Test Case 2:
balance = 999999
annualInterestRate = 0.18

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

```

The code you paste into the following box **should not** specify the values for the variables `balance` or `annualInterestRate` - our test code will define those values before testing your submission.

```

9      b = (b - payment) * (1 + annualInterestRate / 12.0)
10
11  while abs(b) > 0.01:
12      if b < 0:
13          high = payment
14      else:
15          low = payment
16
17      payment = (high + low) / 2.0
18      b = balance
19      for month in range(1, 13):
20          b = (b - payment) * (1 + annualInterestRate / 12.0)
21
22  print "Lowest Payment:", round(payment, 2)
23 v= Low_Payment()
24 print v

```

Correct

## Test results

**CORRECT**

[See full output](#)

[See full output](#)

### Note:

Depending on where, and how frequently, you round during this function, your answers may be off a few cents in either direction. Try rounding as few times as possible in order to increase the accuracy of your result.

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

*You have used 5 of 30 submissions*

---

### If you believe you have correct code but it is marked incorrect after clicking "Check"...

After you submit your code, you can see every test case the graders runs on your code. They compare what your code outputs with what our answer code is supposed to output. Click the small link titled "See Full Output" below the Test Results header.

### "Staff Debug: L397 Error" means your code has an infinite loop

Clicking Check may give you the error:

```
"There was a problem running your solution (Staff debug: L379). We couldn't run your solution (Staff debug: L397)."
```

This means your code is taking too long or has an infinite loop. Test your code with more unique test cases, such as very large or very small values.

---

---

---

© All Rights Reserved



[About](#) [Blog](#) [News](#) [FAQs](#) [Contact](#) [Jobs](#) [Donate](#) [Sitemap](#)

[Terms of Service & Honor Code](#) [Privacy Policy](#) [Accessibility Policy](#)

© edX Inc. All rights reserved except where noted. EdX, Open edX and the edX and Open EdX logos are registered trademarks or trademarks of edX Inc.

POWERED BY  
**OPEN**edX



