Problem Set 1

Handed out: Tuesday, February 12, 2013.
Due: 11:59pm, Thursday, February 21, 2013.

Introduction

This problem set will introduce you to using control flow in Python and formulating a computational solution to a problem. You will design and write three simple Python programs, test them, and hand them in. Be sure to read this problem set thoroughly.

Problem 1: Minimally Investing

You have a dream of buying a new car, and your buddy Bernie Madoff comes to you with a great investment opportunity to help you save up enough money. Bernie's investment opportunity gives an astounding 12% annual interest rate. You make a plan to invest a fixed amount towards your new car every month for one year.

Write a program to calculate how much money you could accumulate in one year if you decide to invest with Madoff.
Take as input:

1. annual interest rate as a decimal
2. monthly investment

For each month, print out statements to report the current balance. Finally, print out the amount you paid over the course of the year and the amount you gained through interest.

amount paid = monthly investment * 12
amount earned from interest = final balance – amount paid

Assume that the interest is compounded monthly according to the balance at the start of the month (before the payment for that month is made), so if the annual interest is 12%, you need to apply a 1% interest on the current balance every month.

Test Case 1

```
>>>
Enter the interest rate as a decimal: .12
Enter the fixed monthly investment: 100
Balance after month 1: $100.0
Balance after month 2: $201.0
Balance after month 3: $303.01
Balance after month 4: $406.04
Balance after month 5: $510.1
Balance after month 6: $615.2
Balance after month 7: $721.35
Balance after month 8: $828.57
Balance after month 9: $936.85
Balance after month 10: $1046.22
Balance after month 11: $1156.68
Balance after month 12: $1268.25
Amount paid: $1200.0
Amount earned from interest: $68.25
>>>
```

Test Case 2

You can see from Test Case 1 that Madoff is offering you a great interest rate. Compare this to the result when using the average interest rate on a savings account - about 0.8%.

```
>>>
Enter the interest rate as a decimal: .008
Enter the fixed monthly investment: 100
Balance after month 1: $100.0
Balance after month 2: $200.07
Balance after month 3: $300.2
Balance after month 4: $400.4
Balance after month 5: $500.67
Balance after month 6: $601.0
Balance after month 7: $701.4
Balance after month 8: $801.87
Balance after month 9: $902.4
Balance after month 10: $1003.01
Balance after month 11: $1103.67
Balance after month 12: $1204.41
Amount paid: $1200.0
Amount earned from interest: $4.41
>>>
```

Hints

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

- Retrieve user input.
- Initialize some state variables that you will need to keep track of throughout your code.
- For each month:
    - Compute the new account balance for the end of that month. This requires applying interest and adding the monthly investment to the old account balance.
    - Print the balance for the end of that month.
- Finally, print out the result statement with the total amount paid and the amount earned through interest.

Use these ideas to guide the creation of your code.

Use the `round` function to limit dollar amounts to only two decimal places.

Problem 2: Paying for Your Dream Car in One Year

Now write a program that calculates the minimum fixed monthly investment needed in order to buy your dream car in one year. The monthly investment must be a multiple of $10 and is the same for all months.

Take as input:

1. the price of your dream car
2. interest rate as a decimal

Print out the minimum fixed monthly investment needed and the final account balance.

Test Case 1

```
>>>
Enter the price of your dream car: 200000.0
Enter the annual interest rate as a decimal: .12
RESULT
Minimum monthly investment to buy car in 1 year: $15770.0
Account balance: $200003.07
>>>
```

Test Case 2

```
>>>
Enter the price of your dream car: 25000.0
Enter the annual interest rate as a decimal: .008
RESULT
Minimum monthly investment to buy car in 1 year: $2080.0
Account balance: $25051.72
>>>
```

Hints

For now, assume that monthly investments have to be multiples of $10. Start at a $10 monthly investment and calculate whether you will be able to afford your car after 12 months (taking into account the interest accrued each month). If $10 monthly payments are insufficient to buy your dream car within a year, increase the monthly payment by $10 and repeat.

Problem 3: Using Bisection Search to Make the Program Faster

You'll notice that in problem 2, your monthly investment had to be a multiple of $10. Why did we make it that way? In a separate file, you can try changing the code 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 when your dream car price is very high. How can we make this program faster? We can use bisection search!

We are searching for the smallest monthly investment such that we can afford a car in a year. What is a reasonable upper bound for this value? If there was no interest, the car could be purchased after monthly investments of one-twelfth of its price, so we must pay at most this much each month. One-twelfth of the price of the car is a good upper bound.

What is a good lower bound? There is no way we could invest less than $0 a month, so we can use that as a lower bound on the minimum monthly investment.

Write a program that uses these bounds and bisection search (for more info check out the Wikipedia page here or section 3.3 of the textbook) to find the smallest monthly payment *to the cent* (no more multiples of $10) such that you can afford your dream car by the end of the year. Try it out with large inputs, and notice how fast it is. Produce the output in the same format as you did in problem 2.

Test Case 1

```
>>>
Enter the price of your dream car: 200000.0
Enter the annual interest rate as a decimal: .12
```

```
RESULT
Monthly investment to buy car in 1 year: $15769.76
Account balance: $200000.03
>>>
```

Test Case 2

```
>>>
Enter the price of your dream car: 25000.0
Enter the annual interest rate as a decimal: .008
RESULT
Monthly investment to buy car in 1 year: $2075.71
Account balance: $25000.05
>>>
```

Note: Depending on where and how frequently you round, your answers may be off a few cents in either direction. If the difference is under about 20 cents, don't sweat it.

Hand-In Procedure

1. Save

Save your solution to Problem 1 as `ps1a.py`, Problem 2 as `ps1b.py`, and Problem 3 as `ps1c.py`. *Do not ignore this step or save your files with a different name.*

2. Time and Collaboration Info

At the start of each file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people you collaborated with. For example:

```
# Problem Set 1A
# Name: Jane Lee
# Collaborators: John Doe
# Time Spent: 3:30
# Late Days Used: 1 (only if you're using any)
#
... your code goes here ...
```

3. Submit

To submit a file, upload it to your Stellar workspace. You may upload new versions of each file until the 11:59pm deadline, but anything uploaded after that time will be counted towards your late days, if you have any remaining. If you have no remaining late days, you will receive no credit for a late submission.

To submit a pset with multiple files, you may do one of two things:

1. You may submit a single .zip file that contains all of the requested files.
2. You may submit each file individually through the same Stellar submission page. Be sure that the top of each code file contains a comment with the title of the file you are submitting, eg. `# Problem Set 1A`

Also, after you submit, please be sure to open up your submitted file and double-check you submitted the right thing. Please do not have more than one submission per file. If you wish to resubmit a file you've previously submitted, delete the old file (using the Stellar "delete" link) and then submit the revised copy.