

Ctonheca

about a month ago



Here we go, i would like to share my solutions for PSET2. I will be glad to discuss insights related to them. regards!

PS2.1

```
b=balance
r=annualInterestRate
p=monthlyPaymentRate
def balance(b,r,p,n=12):
    def ubn(b,p):
        return b-b*p
    if n==0:
        return b
    else:
        n-=1
        b=ubn(b,p)+(r/12)*ubn(b,p)
        #print('Month 1 Remaining balance: {0}'.format(str(round(b,2))))
        return balance(b,r,p,n)
print('Remaining balance: {0}'.format(str(round(balance(b,r,p,12),2))))
```

PS2.2

```
increment=10
b=balance
r=annualInterestRate
p=0
def balance(b,r,p,n=12):
    def ubn(b,p):
        return b-p
    if n==0:
        return b
    else:
        n-=1
        b=ubn(b,p)+(r/12)*ubn(b,p)
        return balance(b,r,p,n)
b1=b
while b1>0:
    p+=increment
    b1=round(balance(b,r,p,12),2)
print('Lowest Payment: ',p)
```

PS2.3

```
b=balance
r=annualInterestRate
def balance(b,r,p,n=12):
    def ubn(b,p):
        return b-p
    if n==0:
        return b
    else:
        n-=1
        b=ubn(b,p)+(r/12)*ubn(b,p)
        return balance(b,r,p,n)
epsilon = 0.01
low = b/12
high = (b*(1+r/12)**12)/12
p = (high + low)/2.0
while abs(0 - balance(b,r,p,12)) >= epsilon:
    if balance(b,r,p,12) < 0:</pre>
        high = p
    else:
        low = p
    p = (high + low)/2.0
print('Lowest Payment: ',round(p,2))
```

Nice to see the creation of a function to run the 12 month evaluations although naming it balance could create confusion and the risk of an error sneaking in. Maybe <code>get_new_balance</code> , evaluate_balance or something similar. Its nice to note that using a function avoids the risk of mutating the balance on sequential tests.

posted about a month ago by kiwitrader (Community TA)

@kiwitrader i'v tried to use recursion in order to calculate the updated balance as it was a math function with this characteristics:

```
Bn=Bn-1 + (r/12)*Bn-1.
```

settled the base of the recursion as n=12 (months) and to stop once reached the desired future state. but i wasn't pretty sure if i was able to implement the concept correctly. can you help me to check if it's a "true" recursion or just a function that works for the problem? thanks

posted about a month ago by Ctonheca

•••

```
•••
Its recursion because the loop is over n = 12 down .. just like mine. I just buried the calculation of
the updated balance in the function call and return a conditional expression rather than if else
with two returns.
def evaluate_year(bal, int, pay, num=12):
    return bal if num <= 0 else \
           evaluate\_year(((bal - pay) * (1 + int)), int, pay, num-1)
Its just the recursive equivalent of:
for _ in range(12):
    bal = ((bal - payment) * (1 + annualInterestRate/12))
posted about a month ago by kiwitrader (Community TA)
                                                                                                                        •••
kinda late but instead of using an epsilon you can use
while round(nuBalance, 2) != 0:
posted 12 days ago by amit_hero291b
                                                                                                                        •••
def minPayment(balance, annualInterestRate): """ Parameters ------ balance : float
annualInterestRate: float
Returns minimum monthly payment to pay off in a year
lowerBound = int(balance/12)
upperBound = (balance*(1+annualInterestRate/12)**12)/12
tolerance = 0.1
iterations = 0
monthlyPayment = 1
while monthlyPayment < balance and monthlyPayment > 0:
    iterations += 1
    monthlyPayment = (lowerBound+upperBound)/2
    newBalance = (balance - monthlyPayment)*(1+annualInterestRate/12)
    for i in range(1,12):
        newBalance = (newBalance - monthlyPayment)*(1+annualInterestRate/12)
        print('Balance month ' + str(i) + ' is ' + str(round(newBalance, 2)))
    \hbox{if newBalance} \ \hbox{$>$} \ \hbox{-tolerance and newBalance} \ \hbox{$<$} \ \hbox{tolerance} \hbox{:}
        print('Lowest Payment: ' + str(round(monthlyPayment, 2)))
        print('Iterations: ' + str(iterations*12))
        return
    if newBalance > 0:
        lowerBound = monthlyPayment
    else:
        upperBound = monthlyPayment
posted 5 days ago by SamBriley
Add a comment
biyani_vishal
                                                                                                                       +
about a month ago
```

Here is my solution for PS2.2 through recursive method.. Assuming balance and annualInterestRate are defined..

```
def cal_balance(balance, annualInterestRate, monthlyFixed = 10):
           duration = 1
           unpaid_balance = balance
           monthlyFixed = monthlyFixed - (monthlyFixed%10)
           while duration <= 12:
               unpaid_balance = unpaid_balance - monthlyFixed
               unpaid_balance = round(unpaid_balance + ((annualInterestRate/12.0) *
unpaid_balance), 0)
               duration += 1
           if unpaid balance <= 0:</pre>
               return monthlyFixed
           else:
               monthlyFixed = monthlyFixed + 10
               return cal_balance(balance, annualInterestRate, monthlyFixed)
    monthlyFixed = cal_balance(balance, annualInterestRate)
    print("Lowest Payment:", monthlyFixed)
```

Add a comment

Parker_Chen

about a month ago



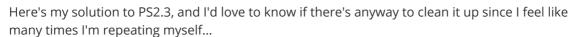
I actually finished the third problem after the deadline because I thought it'll be at midnight, so except the test cases, I have no other ways to check it, but here's my solution. Probably very similar to many others.

```
lower = balance / 12
upper = (balance * (annualInterestRate/12 + 1) ** 12) / 12
pay = (lower + upper) / 2
unpay = balance
while True:
    unpay = balance
    pay = (lower + upper) / 2
    for i in range(12):
        unpay = (unpay - pay) * (1 + annualInterestRate/12)
    if abs(unpay) <= 0.01:</pre>
        break
    elif unpay < -0.01:
        upper = pay
    elif unpay > 0.01:
        lower = pay
print("Lowest Payment: " + str(round(pay, 2)))
```

••• wow, that's so concise! Same here, I just finished it but still wanted to share my solution and compare it to other better ones. def LowestPayment(b, a): b is a positive integer a is a decimal guess is a positive integer Returns lowest fixed monthly payment to clear debt balance = b annualInterestRate = a lowerbound = balance/12 upperbound = (balance *(1+a/12)**12)/12 while balance!= 0: guess = (lowerbound + upperbound)/2updatedbalance = balance for num in range (1, 13): monthlyrate = (annualInterestRate)/12 monthlyunpaid = updatedbalance - guess updatedbalance = monthlyunpaid + (monthlyunpaid*monthlyrate) num += 1 if round(updatedbalance,2) == 0: return round(guess, 2) else: if updatedbalance <= 0:</pre> temp = guess upperbound = temp else: temp = guess lowerbound = temp print("Lowest Payment: ", LowestPayment(999999, 0.18)) I like your solution a lot! posted about a month ago by jaschen2002 Add a comment

helenaliciachu

about a month ago



```
epsilon = 0.01
monthlyInterestRate = annualInterestRate / 12.0
lowerMonthly = balance / 12.0
upperMonthly = balance * (1 + monthlyInterestRate)**12 / 12.0
monthlyPayment = (lowerMonthly + upperMonthly) / 2.0
newBalance = balance
while abs(newBalance) >= epsilon:
    month = 12
    newBalance = balance
    while month > 0:
        unpaidBalance = newBalance - monthlyPayment
        newBalance = unpaidBalance + monthlyInterestRate * unpaidBalance
        month -= 1
    if abs(newBalance) < epsilon:</pre>
        break
    elif newBalance > 0:
        lowerMonthly = monthlyPayment
    else:
        upperMonthly = monthlyPayment
    monthlyPayment = (lowerMonthly + upperMonthly) / 2.0
print('Lowest Payment:', round(monthlyPayment,2))
```

I took the liberty of making some changes, I hope it helps.

```
monthlyInterestRate = annualInterestRate / 12.0
lowerMonthly = balance / 12.0
upperMonthly = balance * (1 + monthlyInterestRate)**12 / 12.0
monthlyPayment = (lowerMonthly + upperMonthly) / 2.0
newBalance = balance
epsilon = 0.01
while abs(newBalance) >= epsilon:
   newBalance = balance
   for i in range(12):
        newBalance -= monthlyPayment
        newBalance += monthlyInterestRate * newBalance
   if abs(newBalance) < epsilon:</pre>
    elif newBalance > 0:
       lowerMonthly = monthlyPayment
   else:
        upperMonthly = monthlyPayment
   monthlyPayment = (lowerMonthly + upperMonthly) / 2.0
print('Lowest Payment:', round(monthlyPayment, 2))
```

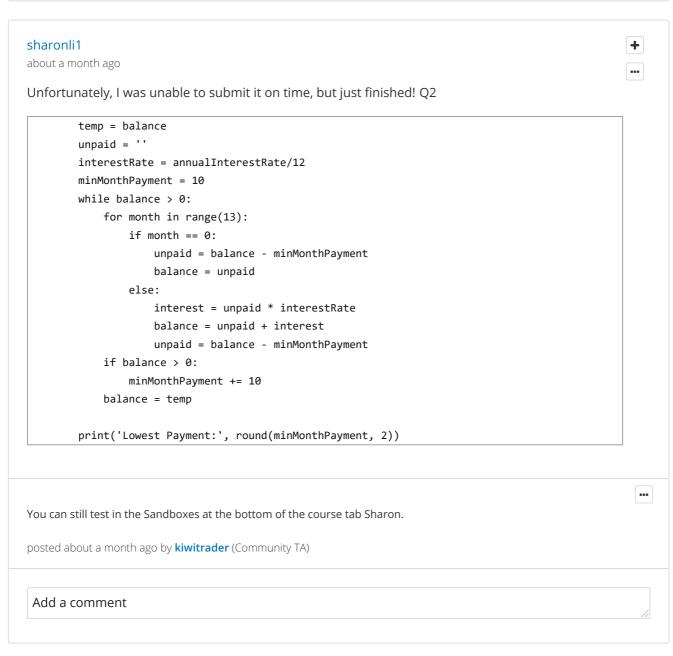
posted about a month ago by wanderson_rb

I like @wanderson_rb's suggestions.

I'd also suggest that your outer while loop would be clearer if you just used True as the condition, since you are checking essentially the opposite condition within the loop and break when it becomes true anyway.

•••

posted about a month ago by c32hedge ... thank you @wanderson_rb and @c32hedge! Much appreciated. posted about a month ago by helenaliciachu Add a comment



richwilliams about a month ago Here's mine for question 3

```
def BalanceAfter1Year(balance, annualInterestRate, payment):
    ''' Returns the balance after 1 year of payments
       Inputs: balance = balance owed,
        annualInterestRate = yearly interest
        payment = monthly payment
       Returns: balanced owed after 12 monthly payments
    for month in range(1,13):
       balance -= payment
       balance += (balance * annualInterestRate / 12)
    return balance
def payIn1Year(balance, annualInterestRate):
    ''' Returns the minimum payment to pay balance in one year
        Inputs: balance = balance owed,
       annualInterestRate = yearly interest
       Returns: Minimum payment rounded to 2 decimals
    # Set initial upper and lower bounds for our guess
    upperPmt = balance * (1+annualInterestRate/12)**12
    lowerPmt = balance / 12
    # Use each guess to reduce the upper and lower bounds
    # until they are .1 cent apart
    while abs( upperPmt - lowerPmt) >= .001:
        # Make a guess in the middle and test it
        payment = (upperPmt + lowerPmt) / 2
        endingBalance = BalanceAfter1Year(balance, annualInterestRate, payment)
        if endingBalance > 0: # Paid too little?
            lowerPmt = payment
                                # Paid too much?
        else:
            upperPmt = payment
    return round(payment,2)
                             # Return it rounded to the penny
# Test it
balance = 999999
annualInterestRate = 0.18
monthlyPayment = payIn1Year(balance, annualInterestRate)
print("Lowest Payment: {0}".format(monthlyPayment))
```

wanderson rb

about a month ago

Here is my solution for PSET 2.3

```
monthly_interest_rate = annualInterestRate / 12.0
epsilon = 0.01
lower_bound = balance / 12.0
upper_bound = (balance * pow(1 + monthly_interest_rate, 12.0)) / 12.0
monthly_payment = (lower_bound + upper_bound) / 2.0
while lower_bound <= upper_bound:
    unpaid_balance = balance
    for i in range(12):
        unpaid_balance -= monthly_payment
        unpaid_balance += monthly_interest_rate * unpaid_balance
    if unpaid_balance == epsilon:
    elif unpaid_balance > epsilon:
        lower_bound = monthly_payment + epsilon
    else:
        upper_bound = monthly_payment - epsilon
    monthly_payment = (lower_bound + upper_bound) / 2.0
print("Lowest Payment: {}".format(round(monthly_payment, 2)))
```

```
Tan_Aileen
                                                                                                    +
about a month ago
payment = 10
while balance >0:
  temptBal = balance
  for i in range (12):
    monthlyUnpaid = temptBal - payment
    newBal = monthlyUnpaid + annualInterestRate/12*monthlyUnpaid
    temptBal = newBal
 if temptBal < 0:</pre>
    break
 payment +=10
print('Lowest Payment: ' + str(payment))
Add a comment
```

```
kiwitrader (Community TA)
```

about a month ago



Here are three recursive solutions. Note the use of conditional expressions to return alternative objects and, if you're not familiar with the form -epsilon <= bal <= epsilon :

PS2-1

```
def evaluate_year(bal, int, pay, num):
    return bal if num <= 0 else \
           evaluate_year((bal * (1 - pay) * (1 + int)), int, pay, num-1)
print('Remaining balance: {:.2f}'.format(evaluate_year(balance, annualInterestRate/12,
monthlyPaymentRate, 12)))
```

PS2-2

```
def find_minimum_payment(balance, interest, payment=0):
    """ Solve with recursion in recursion """
    def evaluate_year(bal, int, pay, num=12):
        return bal if num <= 0 else \
               evaluate_year(((bal - pay) * (1 + int)), int, pay, num-1)
    return payment if evaluate year(balance, interest, payment) <= 0 \
        else find_minimum_payment(balance, interest, payment+10)
print('Lowest Payment:', find_minimum_payment(balance, annualInterestRate/12))
```

PS2-3

```
def find_minimum_payment(balance, interest, lower, upper, epsilon=0.0001):
    def evaluate_year(bal, int, pay, num=12):
        return bal if num <= 0 else \
               evaluate_year(((bal - pay) * (1 + int)), int, pay, num-1)
    bal = evaluate_year(balance, interest, (lower + upper) / 2)
    return (lower + upper) / 2 if -epsilon <= bal <= epsilon else \
        find_minimum_payment(balance, interest, (lower + upper) / 2, upper) if bal > epsilon
else \
        find_minimum_payment(balance, interest, lower, (lower + upper) / 2)
print('Lowest Payment: {:.2f}'.format(find minimum payment(balance, annualInterestRate/12, 0,
balance)))
```

woah!! next level stuff!

I was just able to make a simple recursion. Did not even imagine doing recursion within recursion:) Thanks for sharing!

posted about a month ago by Natasha_C

••• Neat, I didn't know you could do -epsilon <= bal <= epsilon in --that's a common need and the two sides won't always be the same where just using abs would be equivalent. Stylistically, I think the giant return statement in 2-3 is a bit hard to decipher especially with the couple of ternary operators (conditional expressions) thrown in. I think a simple if - elif else with 3 return statements would read better and be about the same amount of work to type:) posted about a month ago by c32hedge ••• Yes. For most Python users that's True and I'd use if elifs in a real world code. I did a bit of Haskell & now Rust both of which have pattern matching so that form reads well to me and I like the way it forces you to include all cases. posted about a month ago by kiwitrader (Community TA) Add a comment



```
Richard_B_UK (Community TA)
                                                                                                        +
about a month ago
                                                                                                        •••
Here are mine:
PS2-1
```

```
unpaidBalance = balance
monthlyInterestRate = annualInterestRate/12
for month in range(12):
    minMonthlyPayment = round(monthlyPaymentRate * unpaidBalance, 2)
    unpaidBalance -= minMonthlyPayment
    unpaidBalance *= (1 + monthlyInterestRate)
    unpaidBalance = round(unpaidBalance, 2)
print("Remaining balance:", unpaidBalance)
```

I rounded inside the loop, reasoning that you can't pay off fractions of a cent.

PS2-2

```
payment = 10 * (0.1 * balance //12)
# it can't be less than this with a positive interest rate
def annualRun(bal, pay):
    :param bal: float to 2 dp, balance to be repaid
    :param pay: payment being tried, int multiple of 10
    :return: int, minimum payment in multiples of 10 that causes bal to be 0 or less
    for month in range(12):
        bal -= pay
        bal *= (1 + annualInterestRate/12)
        bal = round(bal, 2)
    if bal <= 0:
        return int(pay)
    else:
        return annualRun(balance, pay + 10)
print ("Lowest payment:", annualRun(balance, payment))
```

PS2-3

```
startBal = balance
mI = annualInterestRate/12
bottom = balance/12
top = (balance * (1+ mI)**12)/12
while abs(balance) > 0.12:
    balance = startBal
    payment = round((top + bottom) / 2, 2)
    for m in range (12):
        balance = (balance - payment) * (1+mI)
    if balance > 0:
        bottom = payment
    else:
        top = payment
payment = round(payment, 2)
print("Lowest payment:",payment)
```

I actually did this before parts 1 or 2, it was part of the DemoX course that I partially did to get used to the EdX platform.

Add a comment

Dave-UK (Community TA)

break

print('Lowest Payment:' + str(round(guess,2)))

about a month ago

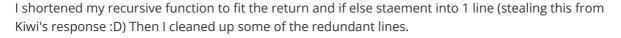
```
My solution for part 3. I've gone for readability over conciseness to hopefully make it a bit easier to
pull apart.
def balanceAfterMonths(amount, months=12):
    b = balance
    monthlyInterestRate = annualInterestRate / 12.0
    for month in range(months):
        monthlyUnpaidBalance = b - amount
        b = monthlyUnpaidBalance + (monthlyInterestRate * monthlyUnpaidBalance)
    return b
low = 0
high = balance
while True:
    guess = (high + low)/2
    result = balanceAfterMonths(guess)
    if abs(result) > 0.01:
        if result > 0 :
            low = guess
        else:
            high = guess
    else:
```



Add a comment

Natasha_C





+

```
def cardpay(due,Interest, n=12):
Input: due is outstanding balance, int or float
Interest is annual interest rate, int or float
n is time period in months
Output: Returns int or float rounded to 2 decimal digits for computing outstanding balance at
the end of n periods, compounded annually
monthInt= Interest/12.0
return due if n<=0 else \
    cardpay((due-guess)*(1+monthInt),Interest, n-1)
low= balance/12.0
high = (balance * (1+annualInterestRate/12.0)**12)/12.0
while True:
    guess = (high+low)/2.0
    check = cardpay(balance,annualInterestRate)
    if -0.01 < check < 0.01: break
    elif check < 0:
        high = guess
    else:
        low = guess
print('Lowest Payment: ', round(guess,2))
```

c32hedge

about a month ago



+

Note: I made a few cleanup changes from my original submissions using the sandbox.

PS 1.1

```
monthlyInterestRate = annualInterestRate / 12.0
for m in range(12):
    minMonthlyPayment = balance * monthlyPaymentRate
    monthlyUnpaidBalance = balance - minMonthlyPayment
    balance = monthlyUnpaidBalance * (1 + monthlyInterestRate)
print("Remaining balance:", round(balance, 2))
```

PS 1.2

```
def findMinPayment(guess):
    b = balance
    for m in range(12):
        b = (b - guess) * (1 + (annualInterestRate / 12))
    if b <= 0:
        return str(guess)
    else:
        return findMinPayment(guess + 10)
print("Lowest payment:", findMinPayment((balance // 120) * 10))
```

PS 1.3

```
monthlyInterestRate = annualInterestRate / 12
minPayment = balance / 12
maxPayment = (balance * ((1 + monthlyInterestRate)**12)) / 12
def findMinPayment(min, max, guess):
    b = balance
    g = round(guess, 2) # use temporary variable to avoid accumulated rounding errors
    epsilon = 1.00
    # If our epsilon is too small there's a chance we will never return
    # even when we've converged to the closest payment. This base case
    # allows us to terminate if that happens. Thanks to @MariaMayskaya
    # for this idea!
    if (max - min) < 0.01: return str(g)</pre>
    for m in range(12):
        b = (b - g) * (1 + monthlyInterestRate)
    if abs(b) <= epsilon:</pre>
        return str(g)
    elif b < -epsilon: # too big
        return findMinPayment(min, guess, (min + guess) / 2)
    else:
                       # too small
        return findMinPayment(guess, max, (guess + max) / 2)
print("Lowest payment:", findMinPayment(minPayment, maxPayment, (minPayment + maxPayment) /
2))
```

Add a comment

Abdelrahman_Hekal

about a month ago

```
Here is my submission for Pset 2 which got 100%:
PS 2-1:
num_months = 12
monthlyInterestRate = annualInterestRate/12.0
for i in range(num_months):
   MonthlyPayment = monthlyPaymentRate*balance
   unpaidBalance = balance - MonthlyPayment
   balance = unpaidBalance + unpaidBalance*monthlyInterestRate
print("Remaining Balance: {}".format(round(balance, 2)))
______
PS 2-2:
num months = 12
monthlyInterestRate = annualInterestRate/12.0
MonthlyPayment = 10 #initial guess
calcBalance = balance
while True:
   for i in range(num_months):
       unpaidBalance = calcBalance - MonthlyPayment
       calcBalance = unpaidBalance + unpaidBalance*monthlyInterestRate
   if calcBalance > 0:
       calcBalance = balance
       MonthlyPayment += 10
   else:
       break
print("Lowest Payment: {}".format(round(MonthlyPayment, 2)))
    .....
PS 2-3:
num months = 12
monthlyInterestRate = annualInterestRate/12.0
calcBalance = balance
upperBound = (balance * (1 + monthlyInterestRate)**12) / 12.0
lowerBound = balance/12.0
guess = 0
while True:
   guess = (upperBound + lowerBound) / 2.0
   for i in range(num_months):
       unpaidBalance = calcBalance - guess
       calcBalance = unpaidBalance + unpaidBalance*monthlyInterestRate
   if calcBalance > 0.01:
       calcBalance = balance
       lowerBound = guess
   elif calcBalance < 0:
       calcBalance = balance
       upperBound = guess
   else:
print("Lowest Payment: {}".format(round(guess, 2)))
```

bbye98

about a month ago





A couple of (nearly) one-liner helper functions that get the job done! Instead of using loops or recursion, I just implemented the closed form solution for the fixed minimum required payment to minimize computation time and maximize performance.

Derivation of closed form solution for fixed minimum monthly payment

Expressions for the monthly balance:

$$b_{0} = b_{0}$$

$$b_{1} = (1 + \frac{r}{12})(b_{0} - p)$$

$$b_{2} = (1 + \frac{r}{12})(b_{1} - p) = ((1 + \frac{r}{12})^{2}(b_{0} - p) - (1 + \frac{r}{12})p) = (1 + \frac{r}{12})^{2}b_{0} - p((1 + \frac{r}{12})^{2} + (1 + \frac{r}{12}))$$

$$b_{3} = (1 + \frac{r}{12})(b_{2} - p) = (1 + \frac{r}{12})((1 + \frac{r}{12})^{2}b_{0} - p((1 + \frac{r}{12})^{2} + (1 + \frac{r}{12})) - p)$$

$$= (1 + \frac{r}{12})^{3}b_{0} - p((1 + \frac{r}{12})^{3} + (1 + \frac{r}{12})^{2} + (1 + \frac{r}{12}))$$

$$...$$

$$b_{12} = (1 + \frac{r}{12})^{12}b_{0} - p((1 + \frac{r}{12})^{12} + (1 + \frac{r}{12})^{11} + ... + (1 + \frac{r}{12})^{2} + (1 + \frac{r}{12})) = 0$$

Expression for calculating the fixed minimum monthly payment:

$$p = rac{{{{(1 + rac{r}{{12}})^{12}}{b_0}}}}{{\sum_{k = 1}^{12} {{{(1 + rac{r}{{12}})^k}}}}$$

Applying the formula for the sum of a geometric series:

$$\sum_{k=1}^{12} \left(1 + \frac{r}{12}\right)^k = \frac{\left(1 + \frac{r}{12}\right)\left(1 - \left(1 + \frac{r}{12}\right)^{12}\right)}{\left(1 - \left(1 + \frac{r}{12}\right)\right)}$$

$$\therefore p = \frac{\left(1 - \left(1 + \frac{r}{12}\right)\right)\left(1 + \frac{r}{12}\right)^{11}b_0}{1 - \left(1 + \frac{r}{12}\right)^{12}}$$

Problem Set 2, Problem 1

```
def remaining(balance, annualInterestRate, monthlyPaymentRate):
    for _ in range(12): balance = (1 + annualInterestRate / 12) * ((1 - monthlyPaymentRate) *
balance)
    print('Remaining balance: %.2f' % balance)
```

Problem Set 2, Problem 2

```
import math
def minPayment(balance, annualInterestRate):
    monthlyInterestRate = 1 + annualInterestRate / 12
    payment = balance * (1 - monthlyInterestRate) * monthlyInterestRate ** 11 / (1 -
monthlyInterestRate ** 12)
    print('Lowest Payment: %.2f' % (math.ceil(payment / (10 **
(math.floor(math.log10(payment)) - 1))) * 10))
```

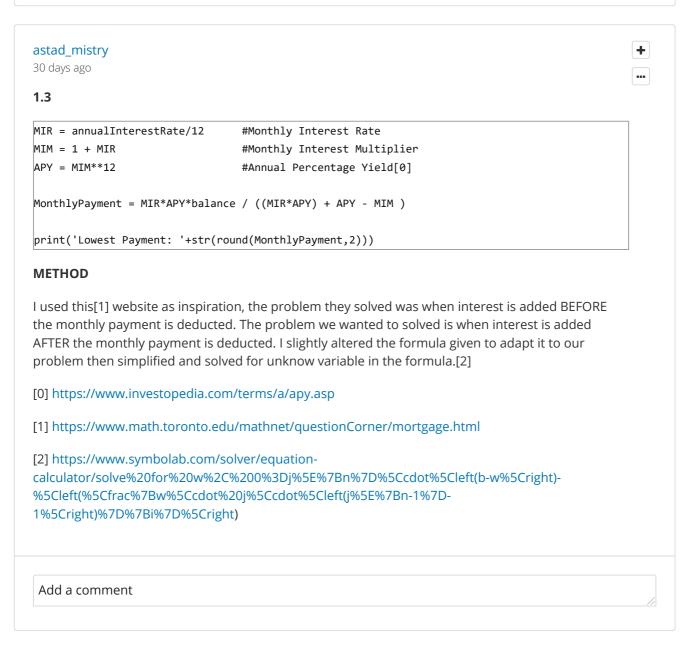
Problem Set 2, Problem 3

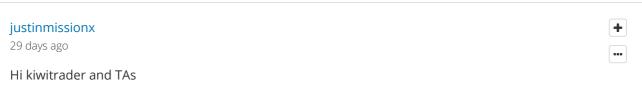
```
import math
def minPayment(balance, annualInterestRate):
    monthlyInterestRate = 1 + annualInterestRate / 12
    print('Lowest Payment: %.2f' % (balance * (1 - monthlyInterestRate) * monthlyInterestRate
** 11 / (1 - monthlyInterestRate ** 12)))
                                                                                                           •••
Hey, you might be interested in my solution. Our solutions are very similar, however I also used
the formula for the sum of a geometric series!
posted 30 days ago by astad_mistry
                                                                                                           •••
Have a look at mine too
posted 12 days ago by Hossein97
Add a comment
```



```
wesamyk1984
30 days ago
hi, i hope someone would check on my code and give feedback thanks in advance
```

```
def Remaining(balance, annualInterestRate, monthlyPaymentRate):
for month in range(1,13):
   min_payment=balance*monthlyPaymentRate
   x=balance-min_payment
                             #unpaid balance
    y=x*annualInterestRate/12
                                  #interests
    balance=round(x+y,2) #new balance
    print("month", month, "balance is:", balance)
Remaining(42, 0.2, 0.04)
Add a comment
```





Missed the deadline on this PSET but would still appreciate feedback if possible. I did notice that my Q3 solution gives lower outputs than Q2 for same input data - presumably just a result of simply a more precise iteration?

These seemed easier than PSET 1 - survived without Thomas' videos(!), but even so, when I look at my answers, they seem remarkably simple - can't believe it seemed so tricky to get there...

Q1:

```
for month in range(12):
    monthlyPayment = balance*monthlyPaymentRate
    balance = balance - monthlyPayment
    balance = balance * (1 + (annualInterestRate/12))
print(round(balance,2))
```

Q2:

```
monthlyPayment = 0
EndBalance = 1
while EndBalance > 0:
    monthlyPayment += 10
    WorkBalance = balance
    for month in range(12):
        WorkBalance = WorkBalance - monthlyPayment
        WorkBalance = WorkBalance * (1 + (annualInterestRate/12))
    EndBalance = WorkBalance
print(str(monthlyPayment))
```

Q3:

```
epsilon = 0.01
monthlyIR = annualInterestRate/12
EndBalance = 1
high = (balance*(1 + monthlyIR)**12)/12
low = balance/12
while abs(EndBalance) >= epsilon:
    monthlyPayment = (high + low)/2
    WorkBalance = balance
    for month in range(12):
        WorkBalance = WorkBalance - monthlyPayment
        WorkBalance = WorkBalance * (1 + (monthlyIR))
    EndBalance = WorkBalance
    if EndBalance > 0:
        low = monthlyPayment
    else:
        high = monthlyPayment
print(str(round(monthlyPayment,2)))
```

Yes, it is that simple. Nicely done.

posted 29 days ago by kiwitrader (Community TA)

BteWierik 27 days ago		•

```
#problem set 2: paying creditcard debt: part 1
def remaining_balance(balance, annualInterestRate, monthlyPaymentRate):
        input: credit card starting balance (balance), annual interest rate
(annualInterestRate) and minimum monthly payments that are done during the year
        output: prints the remaining balance at the end of the first year (rounded in two
decimals)
        remainingBalance = balance
       monthlyInterestRate = annualInterestRate/12
        #each month calculate the remaining balance
        for month in range(1,13):
            #start with the remainingbalance of last month
            previousBalance = remainingBalance
            #make a minimum monlty payment
            minMonthlyPayment = previousBalance*monthlyPaymentRate
            #define the unpaid balance after payment
            MonthlyUnpaidBalance = previousBalance - minMonthlyPayment
            # add the interest due to define the remaining unpaid balance
            remainingBalance = MonthlyUnpaidBalance +
(monthlyInterestRate*MonthlyUnpaidBalance)
            #printstring only used for debugging: prints monthly balances
            #print("Month " + str(month) +" Remaining balance: "
+str(round(remainingBalance,2)))
        #Result at year end
        print("Remaining Balance: "+ str(round(remainingBalance,2)))
remaining balance(balance, annualInterestRate, monthlyPaymentRate
#problem set2 -part 2: find the minimum payment that pays debt in one year
def remaining_balance(balance, annualInterestRate, monthlyPayment):
       input: credit card starting balance (balance), annual interest rate
(annualInterestRate) and monthly payments that are done during the year
       output: prints the remaining balance at the end of the first year (rounded in two
decimals)
        remainingBalance = balance
       monthlyInterestRate = annualInterestRate/12
       #each month calculate the remaining balance
        for month in range(1,13):
            #start with the remainingbalance of last month
            previousBalance = remainingBalance
            #define the unpaid balance after payment
            MonthlyUnpaidBalance = previousBalance - monthlyPayment
            # add the interest due to define the remaining unpaid balance
            remainingBalance = MonthlyUnpaidBalance +
(monthlyInterestRate*MonthlyUnpaidBalance)
            #printstring only used for debugging: prints monthly balances
```

```
#print("Month " + str(month) +" Remaining balance: "
+str(round(remainingBalance,2)))
        #Result at year end
        print("Remaining Balance: "+ str(round(remainingBalance,2)))
        return remainingBalance
#start with a payment of 10
monthlyPayment = 10
#search until the payment is fully done
while remaining_balance(balance,annualInterestRate,monthlyPayment) > 0:
    #if the payment was to low, raise with 10
    monthlyPayment += 10
print('Lowest Payment: ' + str(monthlyPayment))
#problem set 2 - part 3 use bisection search (this one has not been checked yet)
balance = 999999
annualInterestRate = 0.18
def remaining_balance(balance, annualInterestRate, monthlyPayment):
        input: credit card starting balance (balance), annual interest rate
(annualInterestRate) and monthly payments that are done during the year
        output: prints the remaining balance at the end of the first year (rounded in two
decimals)
        remainingBalance = balance
        monthlyInterestRate = annualInterestRate/12
        #each month calculate the remaining balance
        for month in range(1,13):
            #start with the remainingbalance of last month
            previousBalance = remainingBalance
            #define the unpaid balance after payment
            MonthlyUnpaidBalance = previousBalance - monthlyPayment
            # add the interest due to define the remaining unpaid balance
            remainingBalance = MonthlyUnpaidBalance +
(monthlyInterestRate*MonthlyUnpaidBalance)
            #printstring only used for debugging: prints monthly balances
            #print("Month " + str(month) +" Remaining balance: "
+str(round(remainingBalance,2)))
        #Result at year end
        #print("Remaining Balance: "+ str(round(remainingBalance,2)))
        return remainingBalance
epsilon = 0.001
remainingBalance = balance
monthlyInterestRate = annualInterestRate/12.0
```

```
monthlyPaymentLow = balance/12
monthlyPaymentHigh = (balance *(1+monthlyInterestRate)**12)/12.0
monthlyPayment = (monthlyPaymentHigh + monthlyPaymentLow)/2
while abs(remaining_balance(balance,annualInterestRate,monthlyPayment)) >= epsilon:
    #bind output to outerscope
    remainingBalance = remaining_balance(balance,annualInterestRate,monthlyPayment)
    #if the payment was to low, set higher
    if remainingBalance > 0:
        monthlyPaymentLow = monthlyPayment
    #otherwise, if payment was to low, set lower
    else:
        monthlyPaymentHigh = monthlyPayment
    #set the new searchrate
    monthlyPayment = (monthlyPaymentHigh + monthlyPaymentLow)/2
print('Lowest Payment: ' + str(round(monthlyPayment,2)))
```

Mark GD

25 days ago



+

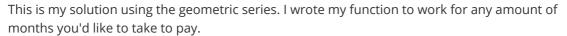
Looking on some advice for slimming down. Brain is fried after PSET3, and I have to really do some reading ahead of the mid-term which I will need to sit Sunday.

P2-3

```
Months = 12
MonIR = annualInterestRate / Months
MinMonPay = balance / Months
MaxMonPay = balance * (1 + MonIR)**Months / Months
RemainBalance = balance
TestMonPay = 1
while RemainBalance > 0.01:
    TestMonPay = (MaxMonPay + MinMonPay) / 2
    for i in range (0, Months):
        RemainBalance -= TestMonPay
        RemainBalance += RemainBalance * MonIR
    if RemainBalance < 0.01 and RemainBalance > -0.01:
    elif RemainBalance > 0:
        MinMonPay = TestMonPay
        RemainBalance = balance
    else:
        MaxMonPay = TestMonPay
        RemainBalance = balance
ReqMonPay = round(TestMonPay,2)
print("Lowest Payment: " +str(ReqMonPay))
I feel like there is some redundancy here. I am wondering whether I could have made use of a while
abs(RemainBalance) > 0.01: would that have meant I could remove the first part of my if-elif-
else?
                                                                                                        •••
Pull all the RemainBalance declarations to first thing under the while so you go from 2 to 1
Make your while condition, if its within the entire range and elif > into and if > 0.1. Which lets
you remove the two lines giving a break.
posted 25 days ago by kiwitrader (Community TA)
Add a comment
```

Hossein97

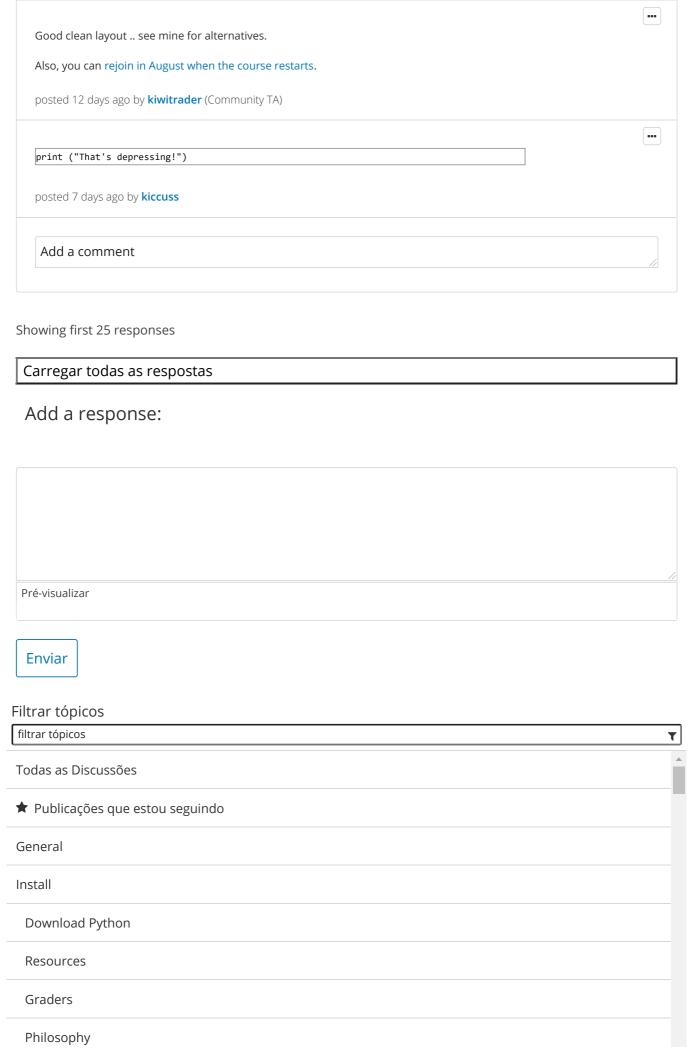
12 days ago



```
def calcPayment(total, interest, months):
    # case1: you want to pay all your balance in 1 month
    if months == 1:
        payment = total
        return payment
    # case2: you want to take multiple months to pay balance
    series = 0
    for i in range(1, months):
        series += (1/interest)**i
    payment = total/(1+series)
    return payment
P = calcPayment(balance, 1+annualInterestRate/12, 12)
print('Lowest Payment:', round(P,2))
```

```
kiccuss
12 days ago
A little late, but moving as far as I can to catch up with the times.
Par 1
def Rbalance(b=balance, m=12, i=annualInterestRate/12, mp=monthlyPaymentRate):
    MinPay = b * mp
   UnPayBal = b - MinPay
    MonthInt = i * UnPayBal
    NewBalance = round(UnPayBal + MonthInt, 2)
    if m <= 0:
        return b
    return (Rbalance(NewBalance, m-1))
print ("Remaining balance: " + str(Rbalance(balance)))
Part 2
def inYearPay(b, pay=0):
    if b <= 0:
        return pay
    pay += 10
    def payBalance(gb):
        for m in range (12):
            gb = ((gb - pay) * (annualInterestRate/12 + 1))
        return gb
```

return inYearPay(payBalance(balance), pay) print ("Lowest payment: " + str(inYearPay(balance)))



ecture	
Video: Global Variables	
Lecture 1	
Exercises 1	
Exercises 2	
Video: Types	



Sobre

edX for Business

Legal

Termos de Serviço e Código de Honra

Política de Privacidade

Política de Assessibilidade

Connect

Blog

Contate-nos

Central de ajuda















© 2020 edX Inc. All rights reserved.

|深圳市恒宇博科技有限公司 <u>粤ICP备17044299号-2</u>