Jordan Lewis
Mr J Lane
SDD Finance Helper
24 June 2016

Task 1

Table of Contents

Compound Interest Calculator

Problem Statement

"Compound interest is the eight wonder of the world. He who understands it, earns it … he who doesn't … pays it." ~ Albert Einstein.

Compound interest with its exponents and rapidly growing nature can be hard to grasp, in particular finding each part (number of periods, rate, etc) given other parameters. There needs to be an intuitive easy to use tool for quickly entering values and having the missing parameters returned.

In addition to this, dates can are annoying to work with. For example if an amount is compounded daily but you know how many months it has been, or vice versa. The program should handle time conversion.

Method of Solution

Input	Process	Output

Principle	adjRate = (interestRate / 100) + 1
	owed = principle * (adjRate ^ periods)
		Amount Loaned: $<principle>

Interest Rate		Interest Rate: <interestRate>%
Number of Periods		Number of Periods: <periods>
		Amount Owed: $<owed>

Input	Process	Output

Amount Owed	adjRate = (interestRate / 100) + 1
	principle = owed * (adjRate ^ -periods)
		Amount Loaned: $<principle>

Interest Rate		Interest Rate: <interestRate>%
Number of Periods		Number of Periods: <periods>
		Amount Owed: $<owed>

Input	Process	Output

Interest Rate	adjRate = (interestRate / 100) + 1
	periods = ceil(ln(owed / principle) / ln(adjRate))
		Amount Loaned: $<principle>

Amount Owed		Interest Rate: <interestRate>%
Principle		Number of Periods: <periods>
		Amount Owed: $<owed>

Input	Process	Output
Principle	rate = ((e ^ (ln(owed / principle) / periods)) - 1) * 100	Amount Loaned: $<principle>
Amount Owed		Interest Rate: <interestRate>%
Number of Periods		Number of Periods: <periods>
		Amount Owed: $<owed>

Algorithm

```
BEGIN InterestCalculator

  BEGIN arrayContains(array, item)
    RETURN array.indexOf(item) != nil
  END arrayContains

  BEGIN printNumberOfPeriods(argDict)
    PRINT “Number of Periods: “ + argDict[“n”]
  END printNumberOfPeriods

  BEGIN printInterestRate(argDict)
    PRINT “Interest Rate: “ + argDict[“r”]
  END printInterestRate

  BEGIN printAmountLoaned(argDict)
    PRINT “Amount Loaned: $“ + round(argDict[“p”], 2)
  END printAmountLoaned

  BEGIN printAmountOwed(argDict)
    PRINT “Amount Owed: $“ + round(argDict[“i”], 2)
  END printAmountOwed

  BEGIN printAllCompound(argDict)
    printAmountLoaned(argDict)
    printInterestRate(argDict)
    printNumberOfPeriods(argDict)
    printAmountOwed(argDict)
  END printAllCompound

  BEGIN printLine()
    line = “”
    FOR i IN 1 TO 50 THEN
      line = line + “–“
    NEXT i
    PRINT line
  END printLine

  BEGIN getRate(argDict)
    RETURN (argDict[“r”] / 100) + 1
  END getRate

  BEGIN calculateInterestRate(argDict)
    amountBorrowed = argDict[“p"]
    amountOwed = argDict["i"]
    numPeriods = floor(argDict["n"])
    RETURN ((e ^ (ln(amountOwed / amountBorrowed) / numPeriods)) - 1) *
    100
  END calculateInterestRate

  BEGIN calculateNumberOfPeriods(argDict)
    amountBorrowed = argDict["p"]
    rate = getRate(argDict)
    amountOwed = argDict["i"]
    RETURN ceil(ln(amountOwed / amountBorrowed) / ln(rate))
  END calculateNumberOfPeriods

  BEGIN calculateAmountBorrowed(argDict)
    amountOwed = argDict["i"]
    rate = getRate(argDict)
    numPeriods = math.floor(argDict["n"])
    RETURN amountOwed * (rate ^ -numPeriods)
```

```
END calculateAmountBorrowed

BEGIN calculateAmountOwed(argDict)
  amountBorrowed = argDict["p"]
  rate = getRate(argDict)
  numPeriods = floor(argDict["n"])
  RETURN amountBorrowed * (rate ^ numPeriods)
END calculateAmountOwed

BEGIN adjustDateFormat(value, currentFormat, targetFormat)
  adjustmentValues = {
    "s": 60.0,
    "m": 60.0,
    "h": 24.0,
    "D": 7.0,
    "W": 2.0,
    "F": 30.0 / 14.0,
    "M": 3.0,
    "Q": 4.0,
    "Y": 1.0,
  }
  keys = "s m h D W F M Q Y".split(" ")
  currentIndex = keys.index(currentFormat)
  targetIndex = keys.index(targetFormat)
  adjustedValue = value
  IF currentIndex < targetIndex THEN
    FOR i IN currentIndex TO targetIndex
      adjustedValue /= adjustmentValues[keys[i]]
    NEXT i
  ELSE IF currentIndex > targetIndex THEN
    ‘ This one decrements
    FOR i IN currentIndex TO targetIndex THEN
      adjustedValue *= adjustmentValues[keys[i - 1]]
    PREVIOUS i
  END IF
  RETURN floor(adjustedValue)
END adjustDateFormat
```

```
BEGIN argDictFromInput(input)
  args = input.replace(" ", “”).split("-")
  dict = {}

  shouldAdjust = nil
  adjustTo = nil

  FOR arg IN args
    IF arg != “" THEN
      try:
        IF arg[0] == “n" THEN
          try:
            dict[arg[0]] = float(arg[1:])
          except:
            dict[arg[0]] = float(arg[1:-1])
            shouldAdjust = arg[-1:]
        ELSE IF arg[0] == “c" THEN
          adjustTo = arg[1:]
        ELSE IF self.arrayContains(self.validArgFlags, arg[0]) THEN
          dict[arg[0]] = float(arg[1:])
        ELSE
          PRINT "Invalid Argument Flag: " + arg[0]
          RETURN None
        END IF
      except:
          PRINT arg[1:] + " is not a valid number."
          RETURN nil
  NEXT arg

  IF dict.length != 3 THEN
    PRINT "Invalid amount of arguments."
    RETURN None
  END IF

  IF shouldAdjust != None AND adjustTo != None THEN
    try:
      dict["n"] = adjustDateFormat(dict["n"], shouldAdjust, adjustTo)
    except:
      PRINT "Invalid time flag, refer to 'help' for all of the valid time
      flags."
      RETURN None
  ELSE IF shouldAdjust != None AND adjustTo == None THEN
    ‘ DO NOTHING: No need to adjust, assuming the user was being verbose
  ELSE IF shouldAdjust == None AND adjustTo != None THEN
    PRINT "Period time format not specified. Add either a d,m,q,y after
    the number to specify."
    RETURN None
  END IF

  RETURN dict
```

```
BEGIN do_compound(line)
  argDict = self.argDictFromInput(line)

  IF argDict == None THEN
    PRINT "Invalid Input"
    RETURN
  END IF

  IF NOT argDict.has_key(“p") THEN
    PRINT "Calculating Principle..."
    argDict["p"] = calculateAmountBorrowed(argDict)
  END IF

  IF NOT argDict.has_key(“r") THEN
    PRINT "Calculating Rate..."
    argDict["r"] = calculateInterestRate(argDict)
  END IF

  IF NOT argDict.has_key(“n") THEN
    PRINT "Calculating Number of Periods..."
    argDict["n"] = calculateNumberOfPeriods(argDict)
  END IF

  IF NOT argDict.has_key(“i") THEN
    PRINT "Caclulating Amount Owed..."
    argDict["i"] = calculateAmountOwed(argDict)
  END IF

  printAllCompound(argDict)
END do_compound

BEGIN do_help(arg):
  PRINT "compound -i <float> -p <float> -r <float> -n
  <float><s,m,h,D,W,F,M,Q,Y> -f <s,m,h,D,W,F,M,Q,Y>"
  PRINT "-i : How much owed at the end."
  PRINT "-p : The principle amount loaned."
  PRINT "-r : The interest rate, in %."
  PRINT "-n : The amount of time."
  PRINT "-c : How often the interest is compounded"
  PRINT "    s : Seconds"
  PRINT "    m : Minutes"
  PRINT "    h : Hours"
  PRINT "    D : Days"
  PRINT "    W : Weeks"
  PRINT "    F : Fortnights"
  PRINT "    M : Months"
  PRINT "    Q : Quarters"
  PRINT "    Y : Years"
  PRINT "Enter 3 of the first 4 flags to find the value of the missing
  one. -f is optional, if it is not used, this will use the amount of
  time as the number of periods."
  PRINT "Type quit to exit the program.”
END do_help

BEGIN do_quit(arg)
  RETURN true
END

BEGIN emptyLine()
END
```

```
  BEGIN do_EOF(arg)
    RETURN true
  END
END InterestCalculator

PRINT "Type 'help' to show the list of flags and how to use this command
line tool.”
InterestCalculator().cmdloop()
```

Test Data

There are no boundaries for non-time unit conversion commands.

Without Unit Conversion	Expected Missing Result
compound -i 259.37 -p 100 -r 10	10
compound -p 100 -r 10 -n 10	259.37
compound -i 259.37 -r 10 -n 10	100
compound -i 259.37 -p 100 -n 10	10

Time unit conversion simple generates a period based on the -n and -c flags, so does not require testing in terms of different combinations of other flags. Also, there are too many possible combinations of units, but the algorithm which converts it is the same.

	With Down Time Unit Conversion	Expected Conversion Result
Bottom Boundary	compound -p 100 -r 10 -n 10h -c s	36000
	compound -p 100 -r 10 -n 10D -c h	240
	compound -p 100 -r 10 -n 10Q -c F	64
Full Traversal	compound -p 100 -r 10 -n 1Y -c s	31536000
Top Boundary	compound -p 100 -r 10 -n 10Y -c M	120

	With Up Time Unit Conversion	Expected Conversion Result
Bottom Boundary	compound -p 100 -r 10 -n 1000s -c m	16
	compound -p 100 -r 10 -n 300h -c D	12
	compound -p 100 -r 10 -n 16F -c Q	2
Full Traversal	compound -p 100 -r 10 -n 31536000s -c Y	1
Top Boundary	compound -p 100 -r 10 -n 1000D -c Y	2

Catching bad inputs. i.e. Negative numbers and when the principle is higher than what is owed. It will also ignore the how often the interest is compounded flag (“c”).

Catching Negatives	Expected Output
compound -i 259.37 -p 100 -r 10	10
compound -p -100 -r 10 -n 10	Cannot have value for argument p be less than 0.
compound -i 259.37 -r -10 -n 10	Cannot have value for argument r be less than 0.
compound -i 259.37 -p 100 -n -10	Cannot have value for argument n be less than 0.
compound -i -259.37 -p -100 r 10	Cannot have value for argument i be less than 0.
compound -p 100 -r -10 -n 1000s -c m	Cannot have value for argument r be less than 0.

Catching Principle > Amount Owed	Expected Output
compound -i 259.37 -p -100 -r 10	10
compound -i 100 -p 259.37 -r 10	Principle cannot be greater than amount owed.
compound -i 100 -p 259.37 -n 10	Principle cannot be greater than amount owed.

Superannuation Calculator

Problem Statement

The nature of having compound interest along with regular deposits all earning a different amount (it's hard to word this) makes it rather difficult to predict how much you will have after some time. There needs to be a easy to use Python tool to calculate or predict how much superannuation an individual should have after a given amount of time.

Method of Solution

Input	Process	Output
Amount of Money in the regular installments	super = money * (((1 + interestRate) ^ periods) - 1) / interestRate	With regular instalments of $<money> at an interest rate of <interestRate>% for <periods>… You have a resulting superannuation of $<super>
Interest Rate		
Number of periods/ instalments		

Algorithm

```
BEGIN
  INPUT money, rate, periods
  ‘ money: The amount of the regular instalments
  ‘ rate: The interest rate earned in the account
  ‘ periods: The number of instalments/times the account is compounded.

  IF money <= 0 OR rate <= 0 OR periods <= 0 THEN
    PRINT “Argument is <= 0. Must be > 0.”
  ELSE
    PRINT "With regular instalments of $" + money + " at an interest rate
    of " + rate + "% for " + periods + " periods…”
    adjRate = rate / 100
    amount = money * (((1 + adjRate) ^ periods) -1) / adjRate
    PRINT "You have a resulting superannuation of $" + round(amount, 2)
  ENDIF
END
```

Test Data

Due to the purely mathematical nature of this program, there only needs to be one test to see if it is indeed correct. Otherwise, there should be error catching such as input validation.

Command	Expected Result
python super.py -m 100 -r 8 -n 20	4576.20
python super.py -m 100a -r 8 -n 20	-m not float
python super.py -m 100 -r 8a -n 20	-r not float
python super.py -m 100 -r 8 -n 20a	-n not int
python super.py -m 100 -r 8 -n 20.1	-n not int
python super.py -m -100 -r 8 -n 20	-m <= 0
python super.py -m 100 -r -8 -n 20	-r <= 0
python super.py -m 100 -r 8 -n -20	-n <= 0
python super.py -m -100 -r -8 -n -20	-m <= 0, -r <= 0, -n <= 0