

Dyalog 2010 Programming Contest

You've decided that it is time to have a home of your own, but you can't decide whether to buy or rent. Renting is easier to understand, but over time you suspect that owning your home might be a better deal. You are lucky enough to have parents who have offered to lend you SOME of the money at a favourable rate, and might be more forgiving of late payments – but you'd still like to explore the problem space in a little more depth. You decide to build a model in APL.

Prizes in this year's contest will be awarded for the best overall solutions to the six programming tasks described in the following. In addition, prizes will be awarded for the three best *presentations* of data arising from working on the problems – without regard to how well the main programming tasks were solved.

The financial model is based upon 12 input parameters:

| Parameter Name | Description | Example Value |
|----------------|--------------------------------|---------------|
| PRICE | Home purchase price | 150,000 |
| COSTS | One-time closing costs | 5,000 |
| TERM | Term of loan (months) | 360 |
| BANKAMT | Amount borrowed from bank | 75,000 |
| BANKRATE | .. at annual mortgage rate (%) | 10 |
| DADAMT | Amount borrowed from parents | 30,000 |
| DADRATE | .. at annual mortgage rate (%) | 5 |
| INFL | Annual inflation rate (%) | 1.5 |
| TAXRATE | Marginal income tax rate (%) | 25 |
| PROPRATE | Annual property tax rate (%) | 1.75 |
| SAVRATE | Savings/investment rate (%) | 3 |
| INITRENT | Initial monthly apartment rent | 600 |

The parameters used in the example should not be taken as predictions of economic conditions in any country - they were simply selected to give an interesting looking curve which ended with buying becoming being more advantageous at the end of the loan term. The output of the model is 15 monthly "time series". In the above example, there would be 360 months of output, the first three months of which are:

| | Series Name | Variable Name | 1st Month | 2nd Month | 3rd Month |
|-----|-----------------------|---------------|------------|------------|------------|
| 1. | After Month | I | 1 | 2 | 3 |
| 2. | Bank Interest | BANKINT | 625.00 | 624.72 | 624.44 |
| 3. | Bank Repayment | BANKPRN | 33.18 | 33.46 | 33.73 |
| 4. | Bank Loan Balance | BANKBAL | 74,966.82 | 74,933.37 | 74,899.63 |
| 5. | Parent Interest | DADINT | 125.00 | 124.85 | 124.70 |
| 6. | Parent Repayment | DADPRN | 36.05 | 36.20 | 36.35 |
| 7. | Parent Loan Balance | DADBAL | 29,963.95 | 29,927.76 | 29,891.41 |
| 8. | Mortgage Tax Savings | TAXSAVE | 156.25 | 156.18 | 156.11 |
| 9. | Property Taxes | PROPTAX | 0.00 | 0.00 | 0.00 |
| 10. | Value of Home | HOMEVAL | 150,186.22 | 150,372.68 | 150,559.36 |
| 11. | Savings While Owning | SAVEOWN | -662.98 | -1,327.26 | -1,992.87 |
| 12. | Home Cashout Value | MECASH | 44,592.47 | 44,184.29 | 43,775.46 |
| 13. | Rent Payment | MORENT | 600.00 | 600.00 | 600.00 |
| 14. | Savings While Renting | SAVRENT | 49,493.75 | 48,986.55 | 48,478.40 |
| 15. | Advantage of Buying | ADVAN | -4,901.28 | -4,802.26 | -4,702.94 |

Formulae

The mathematical formulae used to compute these values are described in this section.

The bank loan is amortized through equal payments over the entire term, using the following formulae:

$$\begin{aligned} \text{BANKMR} &= \text{Bank monthly mortgage rate} \\ &= \text{BANKRATE}/12 \end{aligned}$$

BANKPMT = Bank monthly payment
= $BANKAMT \times BANKMR / (1 - (1 + BANKMR)^{-TERM})$

BANKBAL[0] = Initial bank loan balance
= BANKAMT

BANKINT[I] = Bank mortgage interest in month I
= $BANKBAL[I-1] \times MORATE$

BANKPRN[I] = Bank principal repaid in month I
= $BANKPMT - BANKINT[I]$

BANKBAL[I] = Bank loan balance after month I
= $BANKBAL[I-1] - BANKPRN[I]$

Formulae for the following five items are parallel to the above formulas:

DADMR = Parent monthly mortgage rate

DADPMT = Parent monthly payment

DADBAL = Parent loan balance

DADINT = Parent mortgage interest

DADPRN = Parent principal repaid

The remaining values are computed as follows:

HOMEVAL[0] = Initial value of home

HOMEVAL[I] = Value of home after month I
= $HOMEVAL[I-1] \times ((1 + INFL)^{(1/12)})$
(Home value appreciates at inflation rate)

PROPTAX[I] = Property taxes paid during month I
= $HOMEVAL[I-6] \times PROPRATE/2$
(for months 6, 18, 30, 42,...)
= $HOMEVAL[I-12] \times PROPRATE/2$
(for months 12, 24, 36, 48,...)
(Property taxes are computed annually and paid semiannually)

TAXSAVE[I] = Mortgage tax savings during month I
= $BANKINT[I] \times TAXRATE$
(Interest portion of bank mortgage payments are tax-deductible; payments to parents are not)

SAVEOWN[0] = Initial savings while owning
= 0

SAVEOWN[I] = Savings while owning, through month I
= (SAVEOWN[I-1] x (SAVRATE/12) x (1-TAXRATE))+
TAXSAVE[I] - (PROPTAX[I] + BANKPMT + DADPMT)

HEMECASH[I] = Home cashout value, after month I
= HOMEVAL[I] + SAVEOWN[I] -
(BANKBAL[I] + DADBAL[I])

MORENT[0] = Initial monthly rent payment
= INITRENT

MORENT[I] = Monthly rent payment
= MORENT[I-1] x (1+INFL)
(for months 13, 25, 37, 49,...)
= MORENT[I-1]
(for all other months)
(Rent increases once a year, at inflation rate)

SAVERENT[0] = Initial savings while renting
= (PRICE + COSTS) - (BANKAMT + DADAMT)
(This amount is saved by not buying)

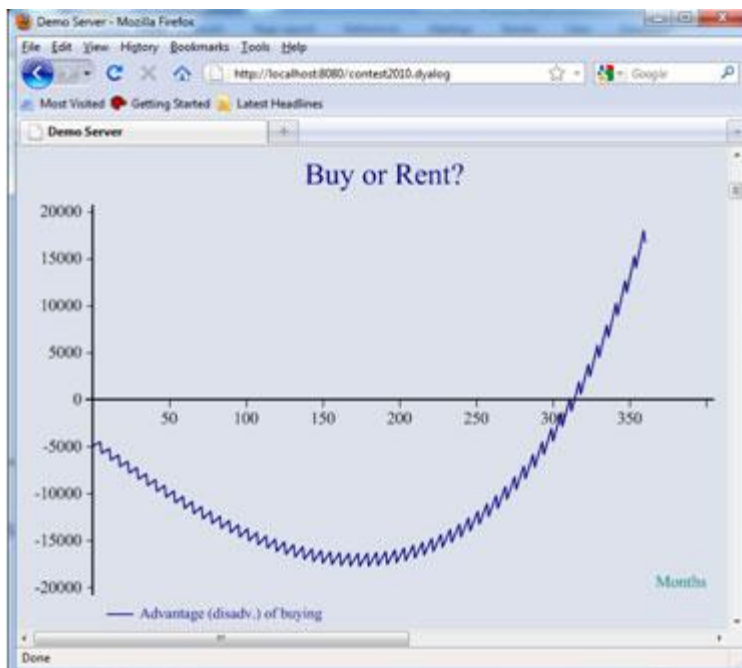
SAVERENT[I] = Savings while renting, through month I
= (SAVERENT[I-1] x (SAVRATE/12) x (1-TAXRATE))
- MORENT

ADVAN[I] = Advantage (disadvantage) of buying
= HEMECASH[I] - SAVERENT[I]

The BuyRentCase XML format

A complete "Buy or Rent Case" can be represented as an XML file containing both the input parameters and the resulting timeseries. A complete **BuyRentCase** file contains both a BuyRentParams element containing a set of input parameters and an element called BuyRentResults with the results ([example of a complete xml file listing](#)). One of the first tasks will be to read input parameters from such a file, which might look like this (if it only contains input parameters and no results):

```
<BuyRentCase>  
  <BuyRentParams HomePrice="150000" ClosingCosts="5000" LoanTerm="360"  
    BankAmount="75000" BankRate="10" DadAmount="30000" DadRate="5"  
    Inflation="1.5" MarginalTax="25" PropertyTax="1.75" SavingsRate="3"  
    InitialRent="600" />  
</BuyRentCase>
```



Graph showing the Advantage of Buying over 360 months.

Your Task

You need to solve as many of the following tasks - 1 to 6 - as possible. To win one of the three main prizes you must solve all of the tasks. It is only necessary to solve two tasks - freely selected from those described on this page and those posed in the [Rosetta Challenge](#), in order to participate in the draw for consolation prizes.

1. Input Processing

Write an APL function called **GetBuyRentParams** which processes XML representing a BuyRentCase and extracts the input parameters. The function should take a character vector containing XML as the right argument, and return a 12-element numeric vector containing the twelve input parameters to the model, in the same order as the table in the introduction.

Hint: The task does not require you to read text files containing XML, but if you would like to store your input data in files, we suggest that you take a look at the workspace FILES (included with Dyalog APL) for tools to read and write files.

2. Base Model Calculation

Write an APL function called **BuyOrRent**, which takes a twelve-element numeric vector of input parameters (which might have been produced by **GetBuyRentParams**) as a right argument, and returns a nested vector with two elements. The first element of the result should be a two-element numeric vector containing the monthly payment amounts to the Bank and Parents, respectively. The second element should be a matrix with one row per month, and 15 columns – one for each of the output timeseries (in the example, the shape of this part of the result should be 360 by 15).

3. Aggregation Function

Write a function named **AggregateSeries**, which *aggregates* the timeseries contained in each column of an input matrix. The right argument should be a matrix (like the second element of the result of **BuyOrRent**). The left argument should be a 2-element vector, where the first element is the number of input periods to be aggregated into each row of output, and the second element is a character vector with one element for each input column, specifying the aggregation function to be used, using one of the following characters: S (sum), A (average), M (maximum), F (first), L (last).

If “mat” is a 4 by 4 matrix containing the integers 1 through 16:

```
      mat
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

then the expression:

```
2 'SAMF' AggregateSeries mat
```

should produce the result:

```
 6  4  7  4
22 12 15 12
```

4. Goal Seeking

Write an APL function **SeekBuyRentParam**, which takes as its right argument a vector of 12 BuyRentParams, where one of the parameters has been replaced by the character '?'. As a left argument, the function will receive a desired value for the "Advantage of Buying" at the end of the loan term. The function should find and return a value for the missing parameter which produces the desired advantage, to within one unit of our currency. If the function is not able to find a suitable value, it should signal a DOMAIN ERROR. For example, if we look at the example problem and would like to know what the initial rent needs to be in order for buying and renting to be exactly equal in terms of the final outcome (Advantage of Buying = 0), we should be able to use this function as follows:

```
0 SeekBuyRentParam 150000 5000 360 75000 10 30000 5 1.5 25 1.75 3 '?'
573.2682766
```

In other words, if the initial rent is lowered to 573, then buying and renting are equally advantageous.

5. Variable Interest

You have agreed with your parents that the interest rate on that loan will remain fixed – and we won't try to compute a net present value of the obligations (dog walking, grandchild

sitting, lawn mowing, snow clearing, and so forth, depending on your climate) that are attached to this loan.

However, the bank offers you the option of a variable interest loan, based on the official rate which is fixed by the central bank, each quarter. At the beginning of each year, your bank will adjust the interest on variable-interest loans to the rate which the central bank has set for the quarter which is just starting. It will re-compute your monthly payments using the standard amortization formula, using the new interest rate and the number of months remaining.

The steering committee in the central bank consists of a bunch of old role players, who have agreed to pursue a fiscal policy based on rolling a single six-sided die. They adjust the interest rate using the number that they get by subtracting 3.5 from the number of pips and dividing by 10, with the limitation that the interest rate cannot go below zero.

Write a function **VariableRisk**, which accepts a set of BuyRentParams on the right (representing the fixed-interest base scenario), and on the left a two-element vector containing the initial interest rate for the variable-interest loan and the number of simulations to run. By running simulations, the function should return a vector of five probabilities (as numbers between 0 and 1), showing the probability that the variable-interest loan will be 1) more than 50,000 more favourable, 2) more than 25,000 but no more than 50,000 more favourable, 3) within 25,000 of the fixed loan, 4) more than 25,000 but no more than 50,000 less favourable, 5) more than 50,000 less favourable than the fixed loan. For example (the following result is not necessarily the "right" answer):

```
9 10000 VariableRisk 150000 5000 360 75000 10 30000 5 1.5 25 1.75 3
600
0.1219 0.5192 0.3582 0.0007 0
```

6. Optimised VariableRisk Function

Write a version of the **VariableRisk** function, called **VariableRiskFast**, which will be judged on how fast it runs on the following problem:

```
8.5 10000 VariableRiskFast 150000 5000 360 75000 10 30000 5 1.5 25 1.75 3 600
```

The *only* criteria used to judge your solution to this task will be the execution speed of the function on the above problem. No subjective criteria will be applied.

Restrictions

The solutions may not make use of any "external" tools or utility libraries not provided as part of a standard Dyalog APL installation.

Submitting Solutions

Solutions should either be submitted as a single UTF-8 file representing a Dyalog namespace containing your solutions - or a single Dyalog workspace containing that namespace. You

may use any APL system to generate the code, so long as your submission is loadable using Dyalog version 12.1.

Judging Criteria

With the exception of the final task, which will be measured exactly, the correct submissions will be evaluated by a panel of judges with considerable experience in the development of APL-based solutions. The criteria used to evaluate your submission include, but are not limited to:

- Did you solve the problem and get correct results?
- Clarity and structure of your solution. How well documented and organized is your solution? Can you pick up the code produced 6 months from now and still understand what it does?
- Use of APL. Does your solution use array-oriented processing where applicable? Solutions that look like C# translated into APL will receive less credit.
- Efficiency of your solution. How fast is your solution? With the exception of the final problem, you should not sacrifice clarity for speed.

The judges' determinations are final and cannot be contested under any circumstances

Presentation Prizes

In order to help visualize the results of the functions that you produce as solutions to the Programming Contest, we have prepared a Web Page which will call many of the functions described above. This solution has been built using the *MildServer*, which is a simple web server written entirely in Dyalog APL (in other words, you do not need to be running Apache, IIS or any other web server in order to use it). You can download and install the MildServer from the APL Wiki, [see http://aplwiki.com/MildServer](http://aplwiki.com/MildServer).

Once the MildServer is installed, all you need to do to integrate your solutions with it is to copy the file `Contest2010sample.dyalog` (which you **can download from the page containing [sample submissions](#)**) to the "Demo" folder of your MildServer installation, and then replace the empty shell functions that it contains with any solutions that you have coded, in order to get the web page to display computed values. The functions that need to be replaced are those which follow the comment which reads:

--- Overwrite the following functions with your own solutions ---

If you have correctly installed the MildServer and the sample page, and direct your web browser to the address <http://localhost:8080/Contest2010>, you should see a page similar to the following (you will need to select one or two data items in order to get a graph):

Dyalog Programming Contest 2010

| | | | | |
|---------------|---------------|-----------------|------|------------------------------------|
| Loan Term: | 360 | Initial Rent: | 600 | <input type="button" value="Run"/> |
| Home Price: | 150000 | Closing Costs: | 5000 | |
| Inflation: | 1.5 | Savings Rate: | 3 | |
| Marginal Tax: | 25 | Property Tax: | 1.75 | |
| Lender | Amount | Interest | | |
| Bank | 75000 | 10 | | |
| Parents | 30000 | 5 | | |

Or upload XML file:

[Download Results as XML](#)

Display Chart for Selected Columns:

— Advantage (disadv.) of buying ▾

— Parent loan balance ▾



Display Report for Selected Columns Display Monthly Details

| | | |
|--|---|---|
| <input checked="" type="checkbox"/> Bank mortgage interest | <input checked="" type="checkbox"/> Bank principal repaid | <input checked="" type="checkbox"/> Bank loan balance |
| <input checked="" type="checkbox"/> Parent mortgage interest | <input checked="" type="checkbox"/> Parent principal repaid | <input checked="" type="checkbox"/> Parent loan balance |
| <input checked="" type="checkbox"/> Mortgage tax savings | <input type="checkbox"/> Property taxes | <input checked="" type="checkbox"/> Value of home |
| <input type="checkbox"/> Savings while owning | <input checked="" type="checkbox"/> Home cashout value | <input checked="" type="checkbox"/> Rent payment |
| <input type="checkbox"/> Savings while renting | <input checked="" type="checkbox"/> Advantage (disadv.) of buying | |

Monthly payments: Bank 658.18, Parents 161.05

[Download Results as XML](#)

| Year | Bank mortgage interest | Bank principal repaid | Bank loan balance | Parent mortgage interest | Parent principal repaid | Parent loan balance | Mortgage tax savings | Value of home | Home cashout value | Rent payment | Advantage (disadv.) of buying |
|------|------------------------|-----------------------|-------------------|--------------------------|-------------------------|---------------------|----------------------|---------------|--------------------|--------------|-------------------------------|
| 1 | 7,481.24 | 416.91 | 74,583.09 | 1,489.95 | 442.61 | 29,557.39 | 1,870.31 | 152,250.00 | 37,426.70 | 7,200.00 | (62,765.67) |
| 2 | 7,437.58 | 460.56 | 74,122.53 | 1,467.30 | 465.25 | 29,092.14 | 1,859.39 | 154,533.75 | 29,659.98 | 7,308.00 | (79,535.38) |
| 3 | 7,389.35 | 508.79 | 73,613.73 | 1,443.50 | 489.06 | 28,603.08 | 1,847.34 | 156,851.76 | 21,697.64 | 7,417.62 | (95,608.04) |
| 4 | 7,336.08 | 562.07 | 73,051.67 | 1,418.48 | 514.08 | 28,089.00 | 1,834.02 | 159,204.53 | 13,537.72 | 7,528.88 | (110,919.72) |
| 5 | 7,277.22 | 620.93 | 72,430.74 | 1,392.18 | 540.38 | 27,548.62 | 1,819.30 | 161,592.60 | 5,178.53 | 7,641.82 | (125,401.23) |
| 6 | 7,212.20 | 685.94 | 71,744.80 | 1,364.53 | 568.03 | 26,980.59 | 1,803.05 | 164,016.49 | (3,381.34) | 7,756.44 | (138,977.73) |

To enter the competition for the Best Presentation, you need to submit code which presents any of the data that you encountered while solving (or merely investigating) the other tasks. The judges will select the three best presentations based on how well they feel each presentation illuminates some aspect of the problem space described in the tasks – WITHOUT regard to what the code which produces it looks like, how efficient it is, or (within reason) how correct the presented data is.

The preferred form of solution is a MildServer page (which could be derived from the sample). At Dyalog's discretion, selected MildServer solutions will be displayed on Dyalog's web server after the closing date for submissions (credit will be given to the authors).

Alternatively, you may add a niladic APL function called **Presentation** to your submitted namespace. This function should produce some kind of visual presentation when executed. The MildServer example uses a charting tool called [SharpPlot](#), which is included with Dyalog APL, but requires the Microsoft.Net framework. You may use any presentation tool that you wish, but you may not use code written in any language other than Dyalog APL to generate the presentation. For example, if you use Microsoft Excel, you must perform *ALL* the automation required to produce the presentation from APL; you may not use scripts internal to Excel. If you use a tool which is not readily available to the judges, you are less likely to win, but if you include screen shots, we will attempt to judge your solution based on these and a reading of your code.

Rosetta Challenge

The Rosetta Code page at <http://www.rosettacode.org> contains solutions to 400 programming tasks, implemented in a wide variety of programming languages. Currently, only a few of the tasks have APL solutions. We have selected five tasks which look as if they might be fun to work on:

- [Animate a pendulum](#)
- [Knapsack problem](#)
- [Happy numbers](#)
- [Hofstadter-Conway \\$10,000 sequence](#)
- [Monty Hall problem](#)

For each task, there is a prize for the best solution.

The five Rosetta Challenge tasks are completely independent of the main Dyalog Programming Contest: you do not need to submit solutions to the main contest tasks in order to participate in the Rosetta Challenge.

Your Task

For each of the following Rosetta Code tasks, you must write an APL function which works as described in the following task descriptions. It is only necessary to solve two tasks - freely

selected from those described on this page and those posed in [Dyalog 2010 Programming Contest](#) in order to participate in the draw for consolation prizes.

1. Animate a Pendulum

The task is introduced on the page http://rosettacode.org/wiki/Animate_a_pendulum. You must write an APL function called **AnimatePendulum**, which takes as its right argument a vector of six numbers:

- [1] Time interval in seconds between successive "frames"
- [2] Starting angle in degrees (rotating clockwise from vertical)
- [3] Length of pendulum in metres
- [4] Gravitational constant in m/s^2
- [5] Initial velocity in m/s
- [6] Number of seconds to run the simulation (0=no limit)

The function should produce an animated graphical representation of a swinging pendulum, in some form of window, using **ANY** technique available from Dyalog APL (anything from "ASCII art" via native Dyalog GUI to XAML/WPF is acceptable).

In addition, the function should return a vector containing the angle of rotation for the first 50 "frames", in order to allow automated verification.

2. Knapsack Problem

Write an APL function called **Knapsack**, which takes a left argument which is a maximum weight, and a right argument which is a matrix with three columns containing item names, weights (in the same units as the left argument) and values (in other words, data corresponding to the table at the top of the page http://rosettacode.org/wiki/Knapsack_problem/0-1). The function should return a vector of item names (selected from the first column), for those items which should be selected to go in the knapsack.

3. Happy Numbers

Write an APL function called **HappyNumbers**, which returns a vector containing the first 8 happy numbers. See http://rosettacode.org/wiki/Happy_numbers for more details.

4. Hofstadter-Conway \$10,000 Sequence

Write an APL function named **HC10k**, which returns a 20-element vector containing maxima of $a(n)/n$ between successive powers of two up to 2^{20} , as described on the page [http://rosettacode.org/wiki/Hofstadter-Conway_\\$10,000_sequence](http://rosettacode.org/wiki/Hofstadter-Conway_$10,000_sequence).

5. Monty Hall Problem

Write an APL function **MontyHallSim**, which takes as its right argument the number of simulations to run, and returns a 2-element vector containing the probability of winning if you "stay", and the probability of winning if you "switch". You can find a description of the Monty Hall problem at http://rosettacode.org/wiki/Monty_Hall_problem.