Some APL Examples

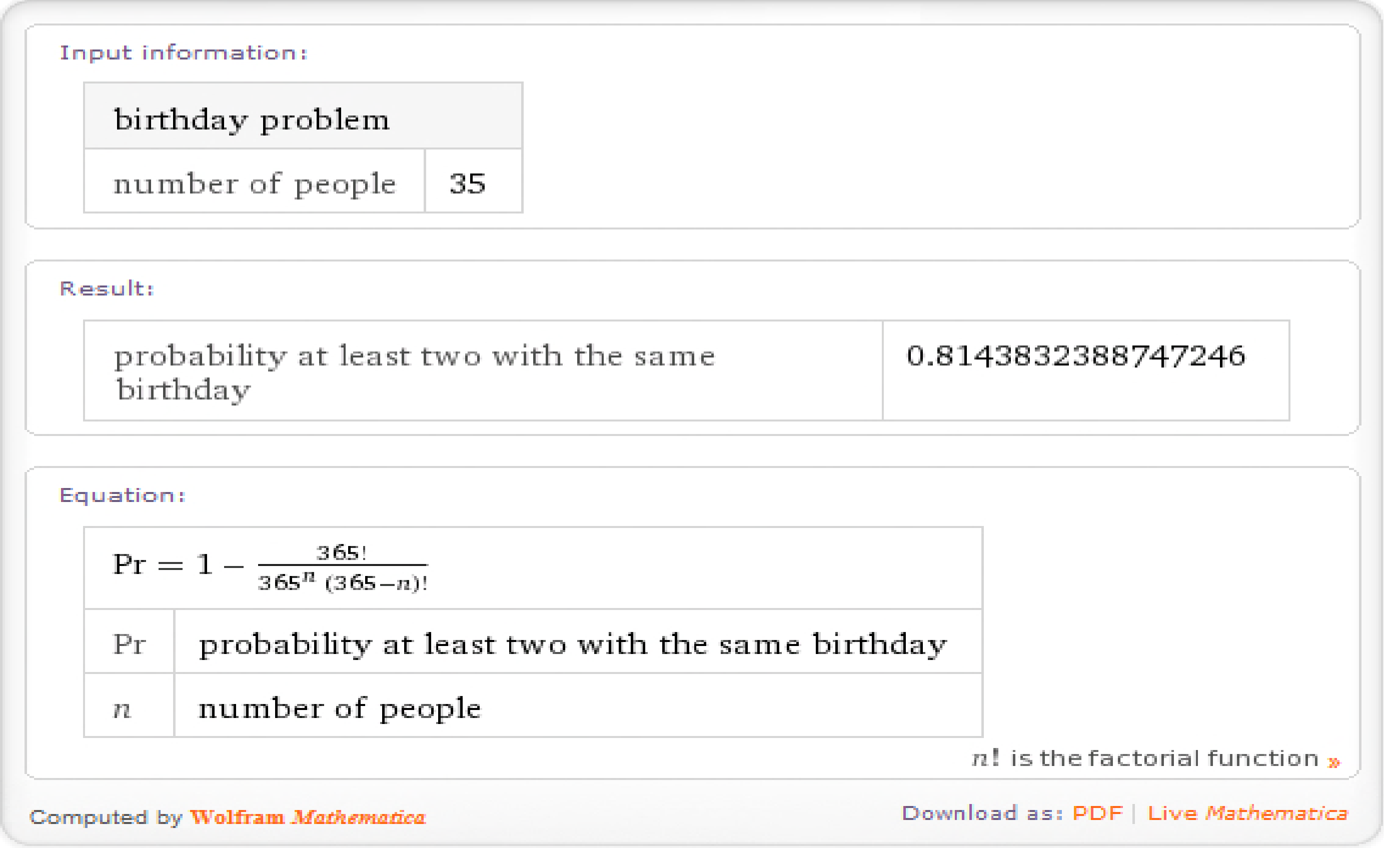
By Jerry M Brennan PhD jbrennan@hawaii.rr.com (808)538-0343

This PDF, all examples & most programs & more available to try at my website:
jerrymbrennan.com (at bottom choose: APL Lessons & Examples: Online Tutorials)

TABLE OF CONTENTS (*=easy *****=hard)

The Birthday Problem *

If you go to a party and there are 35 people there what is the chance that two of the people will have the same birthday.
From Wolfram: The odds are about 81%. The formula is listed below.
http://www.wolframalpha.com/input/?i=birthday+problem+35+people



Input information:

birthday problem	
number of people	35

Result:

probability at least two with the same birthday	0.8143832388747246

Equation:

$$\Pr = 1 - \frac{365!}{365^n\,(365-n)!}$$

Pr	probability at least two with the same birthday
n	number of people

$n!$ is the factorial function »

Computed by Wolfram Mathematica
Download as: PDF | Live Mathematica

In APL you can easily create a program to calculate the formula like this:

```
      birthdaysame←{⎕FR←1287 ⋄ 1-(!365)÷(365*⍵)×(!365-⍵)}
```

The ⎕FR←1287 tells APL use double precision arithmetic (needed because of very large factorial & power calculations). The ⍵ stands for n in the above equation i.e. # people at party. In APL factorial symbol(!) goes in front of number. Also in APL calculation goes from right to left so the entire denominator is calculated first, then the division occurs and finally the subtraction from 1. All to right of ⍝ is a comment & not executed.
Now lets test out the program for the same 35 people at the party.

```
      birthdaysame 35      ⍝ so you enter this for 35 people like above
0.8143832389               ⍝ & computer returns .81438 same result as above
```

So there is about an 81% chance that two people will have the same birthday. Lets try a couple of others and see the percents.

```
      birthdaysame 25      ⍝ you enter this for 25 people at the party
0.568699704                ⍝ get ~57% of time at least 2 have same birthday
      birthdaysame¨ 50 66 ⍝ enter this get odds for each(¨) 50 & 66 people
0.9703735796 0.9980957046 ⍝ 97% for 50 people and 99.8% for 66 people.
```

So it looks like once we get to about 66 people odds are almost 100%.

NOW LETS PLOT THESE PROBABILITIES **

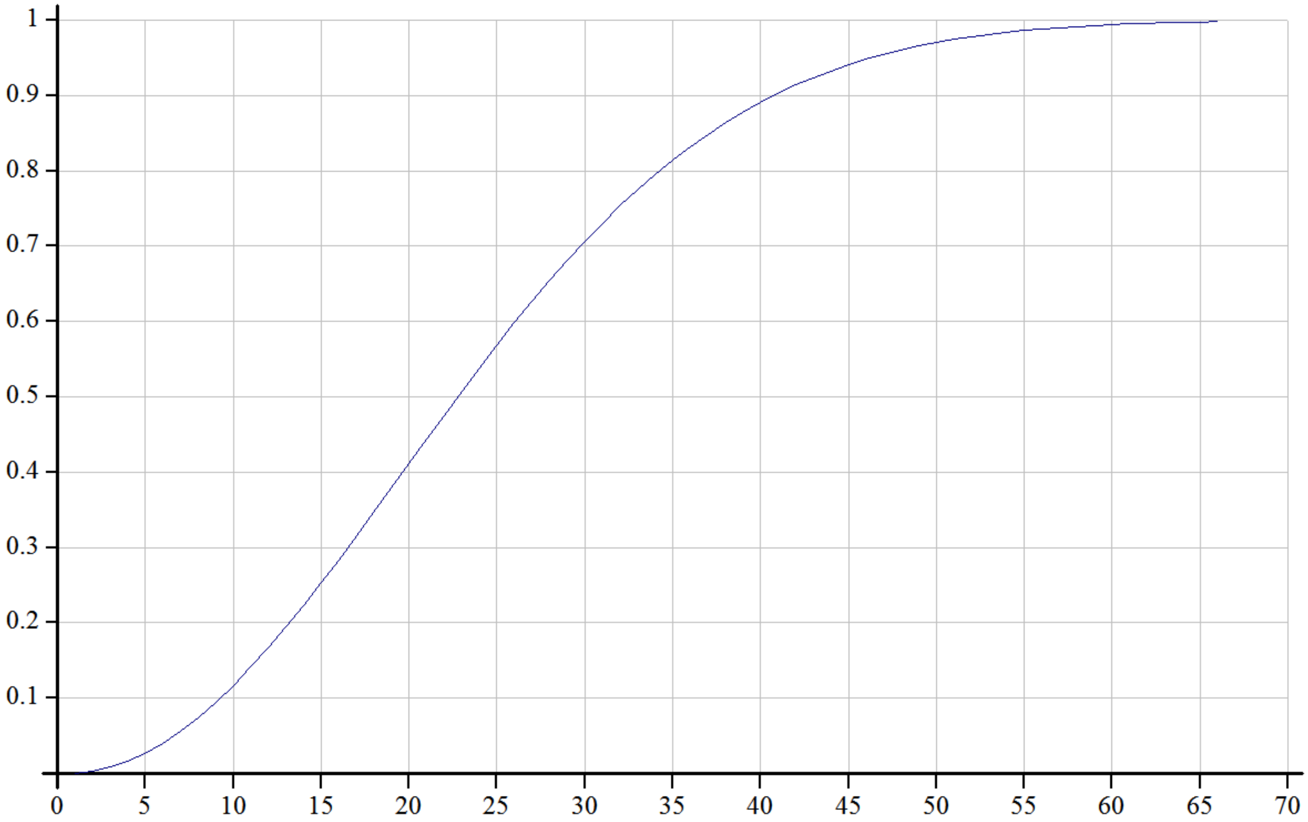
for all the #'s of people from 1 to 66. Apl has a special operator called iota (⍳) that will easily generate all the numbers for one to any number you want.

```
      ⍳6
1 2 3 4 5 6              ⍝ monadic ⍳ called: index generator makes numbers 1-6
      10+⍳8
11 12 13 14 15 16 17 18   ⍝ generates numbers 1-8 first then adds 10 to each. So:
```

So here's code line that calculates/plots odds each(") # of people from 1 to 66.

```
plotxy X (Y←birthdaysame ¨X←⍳66) ⍝ for each # 1 to 66(⍳66) and plot
View PG ⍝ to see it          ⍝ press enter on this line to see plot
```

APL has very sophisticated plotting/graphing & with a little effort we can make a grid line plot. (Y **axis**:the odds for # 1-66 by X **axis**:the # 1-66) You can see below for example that for 40 people the odds is about 90%. Plotting all possible odd shows a curve not a straight line.



Here's the plot fns : To create it type **)ed plotxy** press enter and type in

```
R←{ax0}plotxy data
⍝ plot data:x=col1 y=col2 or x=vector1 y=vector2
 ax0←0=⎕NC'ax0' ⍝ if no ax0 axes cross at 0
 :If 2=≡data ⋄ data←⍉↑data ⋄ :End
 ch.Set'Lines' 1 2 4 5
 ch.Set¨(ax0,ax0,1)/('Xint' 0)('Yint' 0)('XYPLOT,GRID')
 ch.Plot data ⋄ PG←ch.Close
 R←'View PG ⍝ to see it'
```

Press ESC when the above lines have been entered and then copy in rainpro.

```
      )copy rainpro     ⍝ this will copy in all the fancy APL graphics
```

Two Dice – How Lucky Are You? **

In APL the ? is used to generate random numbers so

```
      ?6  ⍝ generates a random number between 1 and 6 each time you do it
3         ⍝ got a 3 this time
      ?6
5         ⍝ got a 5 this time
```

To throw two dice you need two 6's

```
      ?6 6
2 4       ⍝ got a 2 and a 4
```

```
dice←{⍝ Here's a program to interpret 2 dice throws. To call: dice ?6 6
     ω≡6 6:ω,'Box Cars'      ⍝ if inputs(ω) match(≡)6 6 display Box Cars
     ω≡1 1:ω,'Snake Eyes'    ⍝ if inputs(ω) match(≡)1 1 display Snake Eyes
     =/ω:ω,'Pair'            ⍝ if inputs(ω) are equal(=/) display Pair
     7=+/ω:ω,'Seven'         ⍝ if inputs(ω) sum(+/)=7 display Seven
     ω,'Unlucky'             ⍝ else display Unlucky
 }
     dice ?6 6   ⍝ turns 2 6's into random numbers between 1 and 6
2 5 Seven        ⍝ result was a 2 and 5 which sums to lucky 7
     dice ?6 6   ⍝ try again 2 random numbers between 1 and 6
2 1 Unlucky      ⍝ result this time was 2 and 1 which matches none of if's
     dice¨ ?5⍴⊂6 6 ⍝ 5 sets(5⍴) of 2 6's(⊂6 6), random & check each(¨) set
 2 3 Unlucky  2 2 Pair  6 4 Unlucky  2 3 Unlucky  3 4 Seven ⍝ 5 results
```

Probability of Two Dice Being Equal ***

Lets do 5 throws of 2 dice. To do this enclose(⊂) 5 copies(⍴) of two 6's and let the ? turn all 5 pairs of 6's into random pairs of numbers 1-6:

```
      ?5⍴⊂6 6             ⍝ this is APL command and result is on next line
 6 2  2 1  6 6  6 6  1 4  ⍝ we got five pairs of numbers(notice extra
space between each pair. Also notice we got two pairs (of 6's). To make APL
count matches we put an equal sign(=) between each pair(/¨) like this.
      =/¨?5⍴⊂6 6
0 0 1 1 0        ⍝ The ones tell us which pairs matched: (pairs 3 and 4)
```

Now lets add these 1's(with +/) getting 2 & divide by 5 to get the odds of .4 Finally multiply by 100 to get 40 (for 40% matching pairs)

```
     100×(+/=/¨?5⍴⊂6 6)÷5
40                         ⍝ so this time we got 40% matches (2÷5)
```

Now lets write a program to do this and call it **DiceEqual**.

```
     DiceEqual←{100×(+/=/¨?ω⍴⊂6 6)÷ω}  ⍝ variable omega (ω) replaces 5
```

Now with **ω** we can try bigger samples and see if the real underlying probability is indeed 40%. Lets just go for it with a million throws to get a real good idea what the real probability is.

```
      DiceEqual 1000000     ⍝ throw pair of dice million times get % equal
16.6442        ⍝ looks like about 16.6% of time dice will match (not 40%).
```

Now lets try it 5 times with 100 throws each time(¨):

```
      DiceEqual¨5⍴100
27 26 15 16 21          ⍝ got some variability between 15% and 27% matches
```

Now lets try it 5 times with 1,000,000 throws each time(¨)

```
      DiceEqual¨5⍴1000000
16.6033 16.6032 16.6488 16.6859 16.6377 ⍝ always got 16.60% to 16.69
```

From this we can see the advantage of large random samples. Large samples are less variable and they are more accurate. There are actually formulas that allow us to see the actual odds. The probability of two independent random events occurring together is simply the product of the probabilities of each event. In this case each die has 6 sides so the probability of getting say a 3 on one throw is 1/6 and the probability of any particular pattern such as “3 3” is 1/6×1/6=1/36 which is 1 chance in 36. In our case we have 6 different ways to get a pair 1 1,2 2,3 3,4 4,5 5 and 6 6. So the odds of getting a matching pair is 6/36 which equals .1666666666. Looking back at our 5 1 million throws we can see that a sample size of 1,000,000 produces some pretty accurate results while the 5 size 100 samples were not so good. Just for fun lets try 1,000,000 throws 20 times and average them.

```
      Mean←{+/⍵÷⍴⍵}  ⍝ Mean program add up #’s(+/⍵) and divide by n(⍴⍵)
      Mean¨ (1 2 3)(8 6)(?1000⍴50) ⍝ Mean each(¨)note:last=1000 rand# 1-50
2  7  25.015                       ⍝ means for each group of numbers.
      Mean  DiceEqual¨20⍴1000000   ⍝ 20 groups of 1,000,000 pair throws
0.16662385   ⍝ took 17 seconds for my computer but is even more accurate.
```

Now lets see if the larger samples are less variable as suggested above by looking at some frequency plots. First I need a rounding function to round the percents to whole numbers so they can be put in categories. APL has the floor function(⌊) which is useful here. But we can’t just use the floor function because it always rounds down.

```
      ⌊1.2 3.4 1.8
1 3 1    ⍝ all numbers are rounded down, but we need 1.8 to be rounded up.
```

A solution is to add .5 to each number then use the floor(⌊) function

```
      ⌊.5+1.2 3.4 1.8   ⍝ so the #’s become 1.7 3.9 2.3 and
1 3 2                   ⍝ proper rounding is done. ⌊1.7 3.9 2.3 is 1 3 2
```

So here is my round function. It is a little more general than needed here so it can round to any number of decimal places by multiplying the number by some magnitude of 10, adding .5, finding the floor then dividing it back down by the same order of 10. It also has a default(⍺←0) which says to round to 0 decimal places if nothing else is specified to the left.

```
          round←{⍺←0 ⋄ (⌊0.5+⍵×10*⍺)÷10*⍺} ⍝ define the round function
          round 2345.45678             ⍝ default round to 0 (whole number)
2345
          1  round 2345.45678            ⍝ round to 1 decimal place
2345.5
          2  round 2345.45678            ⍝ round to 2 decimal places
2345.46
          ¯2  round 2345.45678           ⍝ round to 100’s place with ¯2
2300
```

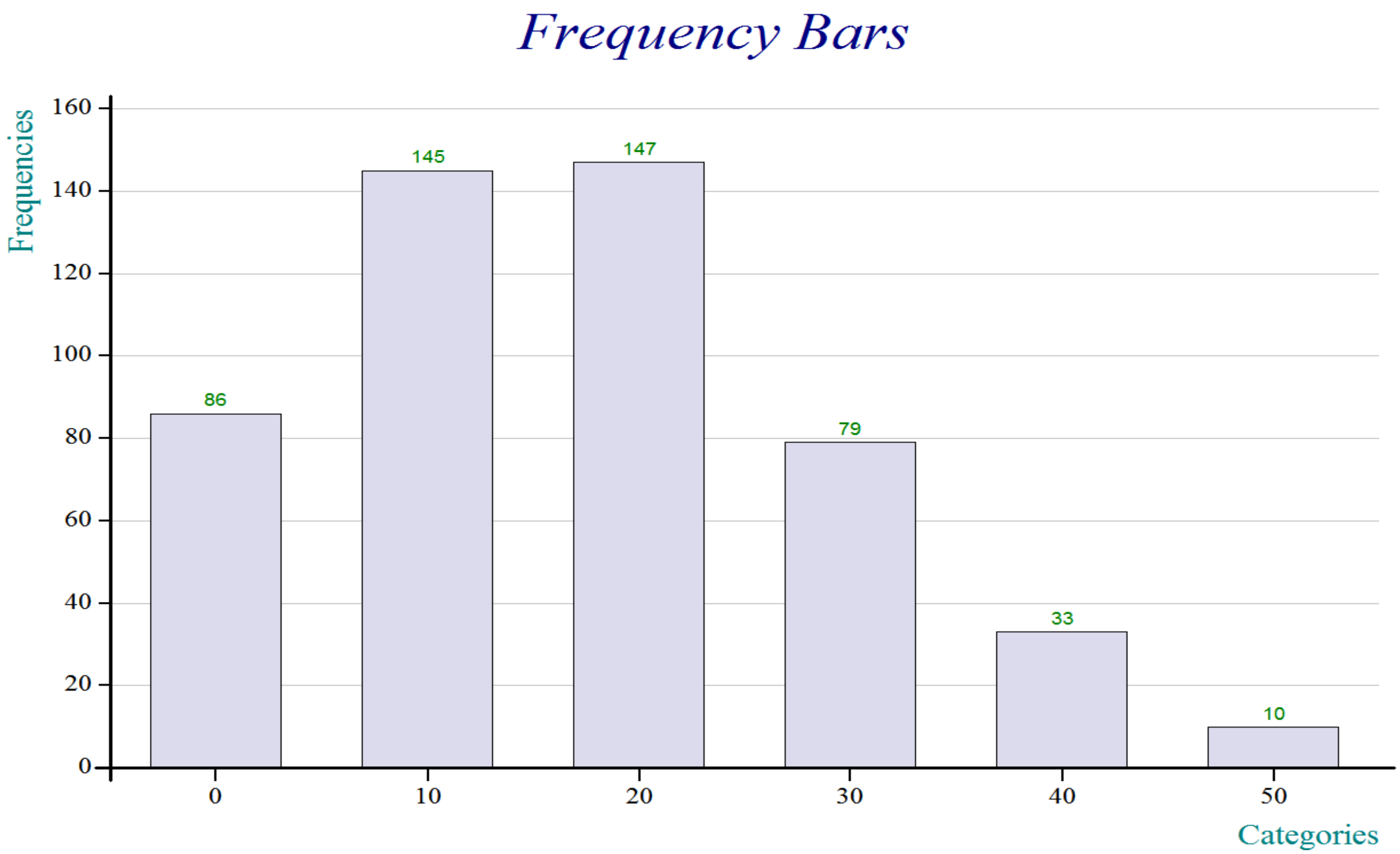
Next we need a program to put rounded results into categories:

```
      Freq←{↑(⍕¨u)(+⌿⍵∘.=u←u[⍋u←∪⍵])}    ⍝ Here is the freq program:
```

Freq finds unique(**u**) input values(**⍵**), sorts them(**u[⍋u]**), makes a table(rows=**⍵** & cols=**u**) where each row value is matched to each col value(∘.=) so each cell is 1 or 0, then adds up all matches in each col(+⌿) to determine the frequencies for each unique #. (+⌿⍵∘.=u)

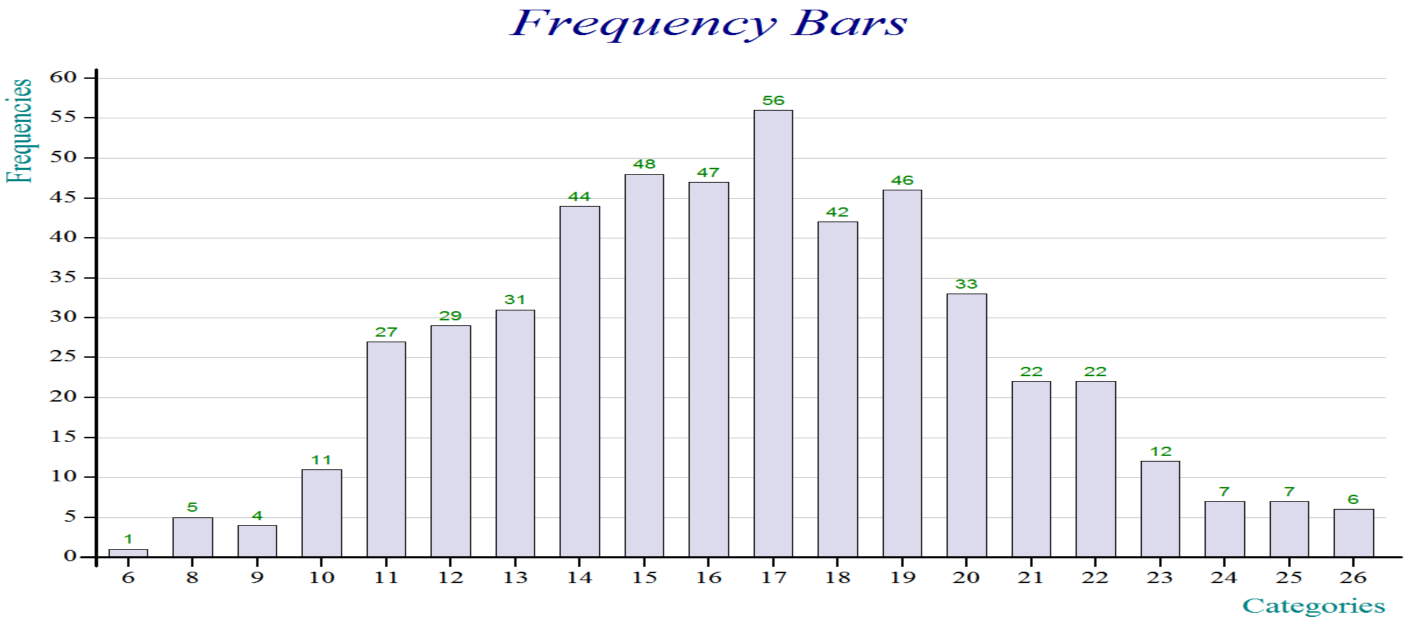
Now we can do some plotting using the built in **barchart icon.** Lets create 500 10’s(**500⍴10**) and send each(¨) to **DiceEqual** which creates 500 random samples of size 10 of 2 dice tosses and calculates percentage of equal pairs for each of the 500 samples of size 10. The percentages are passed to **round** which rounds them to whole numbers and passes them to **freq** which counts up how many times each unique (∪) percentage occurs and creates a table of the values and their frequencies passes this table to **DATA** where the values and their frequencies are stored. The plus sign(**+**) at the beginning of line displays 2 row data table that’s stored in **data**

```
      +DATA←Freq round DiceEqual¨500⍴10  ⍝ call with 500 samples size=10
 0   10   20  30  40  50    ⍝ this row shows the percentages that occurred
86  145  147  79  33  10    ⍝ this row is frequency of percentage above it
      FreqBar DATA          ⍝ Now make a Frequency Bar chart of DATA
```



We have a range of 0% to 50% matching pairs, showing tremendous variability So 0% matches occurred 86 times 10% matches occurred 145 times etc.
Now lets try 500 samples of size 100

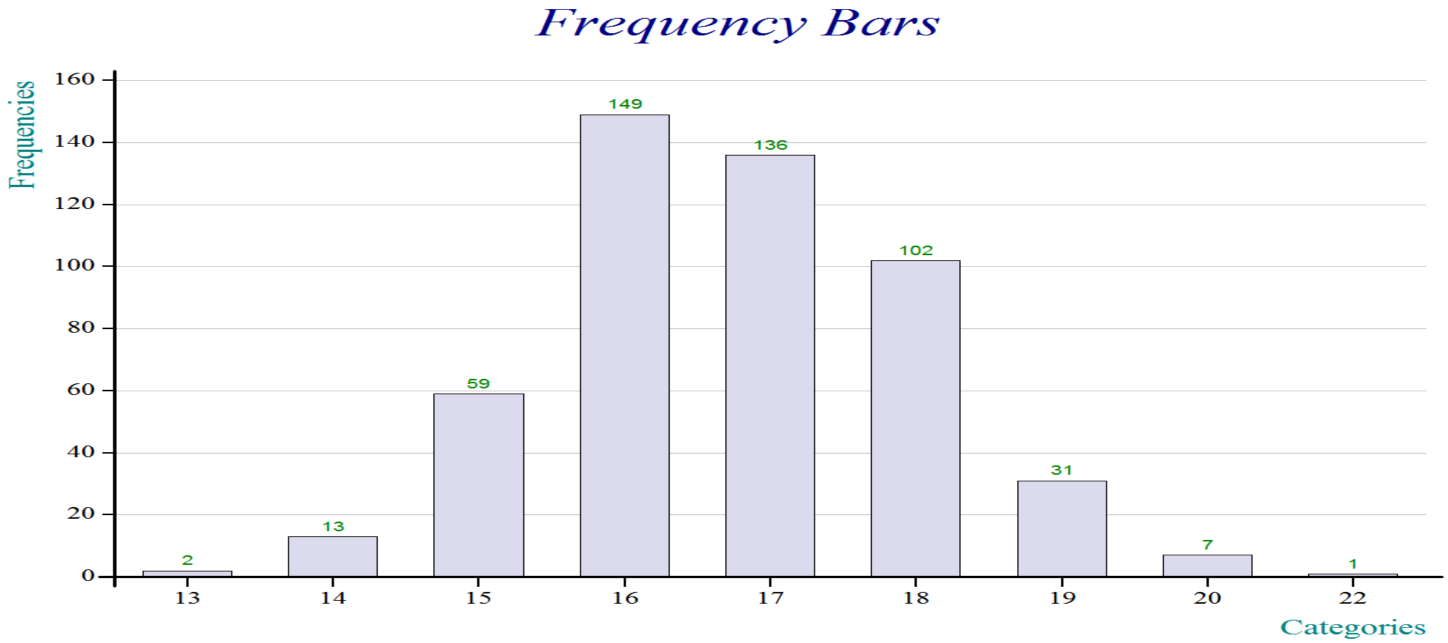
```
      +DATA←Freq round DiceEqual¨500⍴100  ⍝ 500 samples of size 100
 6  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
 1  5  4  11  27  29  31  44  48  47  56  42  46  33  22  22  12   7   7   6
      FreqBar DATA                              ⍝ Frequency Bar chart of DATA
```



A smaller range of 8% to 26% matching pairs but still lots of variability

Lets try 500 samples of 1,000

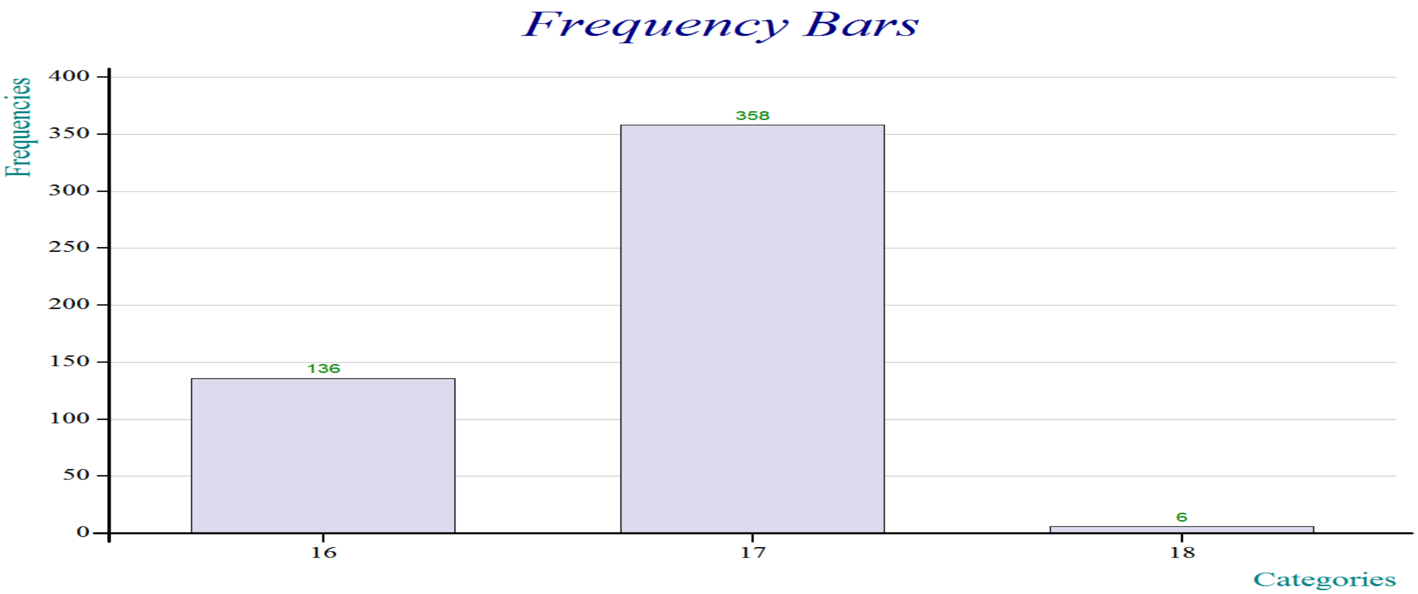
```
      +DATA←Freq round DiceEqual¨500⍴1000
 13  14  15   16   17   18  19  20  22
  2  13  59  149  136  102  31   7   1
      FreqBar DATA                     ⍝ Frequency Bar chart of DATA
```



Even smaller range of only 13% to 22% matching pairs. We are getting close

Lets try 500 samples of 10,000:

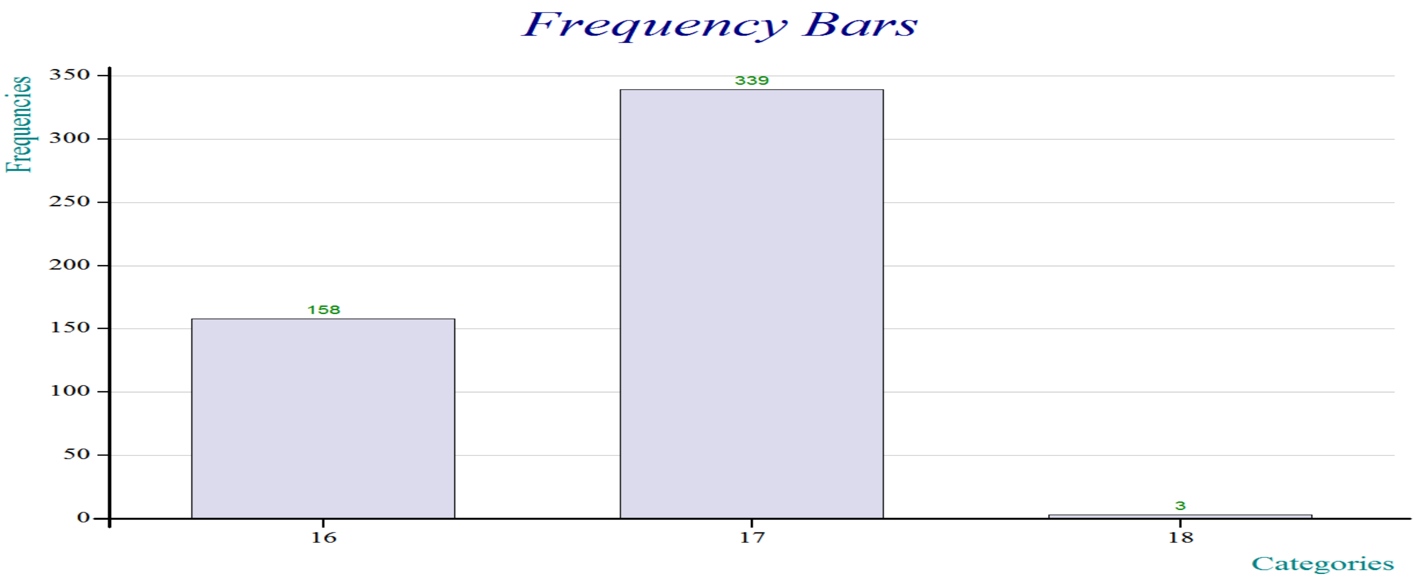
```
      +DATA←Freq round DiceEqual¨500⍴10000 ⍝ 500 samples of size 10,000
   16   17  18
  136  358   6
      FreqBar DATA                     ⍝ Frequency Bar chart of DATA
```



We have a range of only 16% to 18% matching pairs with only 6 at 18 and many more at 17 than 16. Thus we are zeroing in on the theoretical value of 16.66666. A sample size of 10,000 thus almost guarantees a close estimate of the true value. Good scientific research thus tries to get large sample sizes if possible for this reason. Sampling errors becomes a much smaller concern.

Lets try sample size 10,000 again to see if we'll have consistent results:

```
      +DATA←Freq round DiceEqual¨500⍴10000 ⍝ 500 samples of 10,000 again
   16   17  18
  158  339   3
      FreqBar DATA                         ⍝ Frequency Bar chart of DATA
```



We have range of 16% to 18% again and other frequencies are very very close. Replication is another important part of the scientific method in verifying that we are on the right track. Other things we could do to verify this result would be for you to try this on your computer which may have a different random number generator or you could do the 500×10000 dice rolls yourself to check these results. ;)

Name The Order Of The Presidents *

A clueless student faced a pop quiz to match list of 24 US presidents with another list of 24 terms(years) of office. Being clueless they had to guess every time. On average how many would they guess correctly?_____

Since we don't know the probability formula lets run quick Monte Carlo simulations. Use APL random # generator ? to get the avg # you'd get by randomly guessing. First simulate match test with only 5 numbers to match.

```
      5?5                  ⍝ enter this (use 5 not 44 for the moment)
5 4 3 1 2                  ⍝ and the numbers 1-5 are rearranged randomly
      5?5                  ⍝ enter it again
3 4 2 1 5                  ⍝ and get a different order back
      (5?5)=(5?5)          ⍝ compare teachers correct order to your guesses
0 0 1 1 0                  ⍝ and you got 2 right (the 3rd and 4th ones).
      (5?5)=(5?5)          ⍝ try it again
0 0 0 0 0                  ⍝ and you got 0 right
      +/(5?5)=(5?5)        ⍝ lets add them up so we don't have to count
1                          ⍝ we got 1 of the 5 right this time.
```

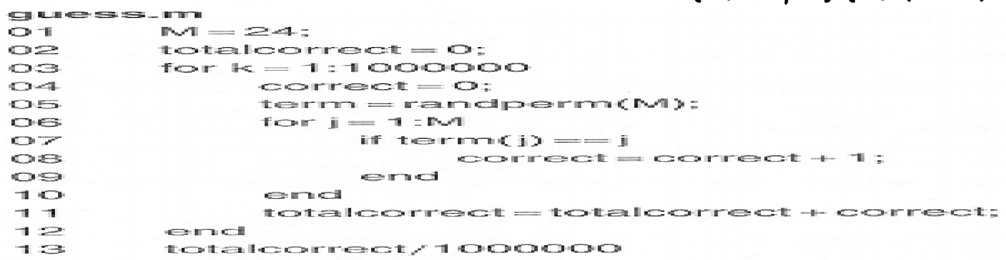
Now turn this to a function & run lots of times to see the average result.

```
avg←{+/⍵÷⍴⍵}                       ⍝ first write fns to compute average
presmatch←{+/(⍵?⍵)=(⍵?⍵)}          ⍝ fns counts # matches for ⍵ presidents
avg presmatch 5                    ⍝ test it for 5 presidents
0                                  ⍝ no matches
      avg presmatch¨100⍴5          ⍝ test 5 pres 100 times using each(¨)
0.95                               ⍝ average correct =.95
      avg presmatch ¨100⍴5         ⍝ average this time =1.11
1.11
Now run 100,000 times & get more accurate estimate then try 44 presidents.
      avg presmatch ¨100000⍴5      ⍝ first for 5 presidents
1.000726                           ⍝ pretty close to 1
      avg presmatch ¨100000⍴24     ⍝ now for the 24 presidents
1.000088                           ⍝ interesting basically 1 again.
      avg presmatch ¨100000⍴125    ⍝ what if there were 125 presidents?
.99986                             ⍝ still ~1 that is pretty unexpected!
```

Conclusion:

1. **Study!** Guessing is not going to get you very far on any matching test.
2. **Learn APL**, so you can easily figure out what risks are in many things.

Above example is from **Digital Dice:Computational Solutions to Practical Probabability Problems by Paul J. Nahin** 2008. The book uses **MATLAB** a fancy math/statistics program to show code for this example. Here is equivalent 13 lines of MATLAB to 1 line of APL: **{+/⍵÷⍴⍵}{+/(⍵?⍵)=⍵?⍵}¨1000000⍴24**



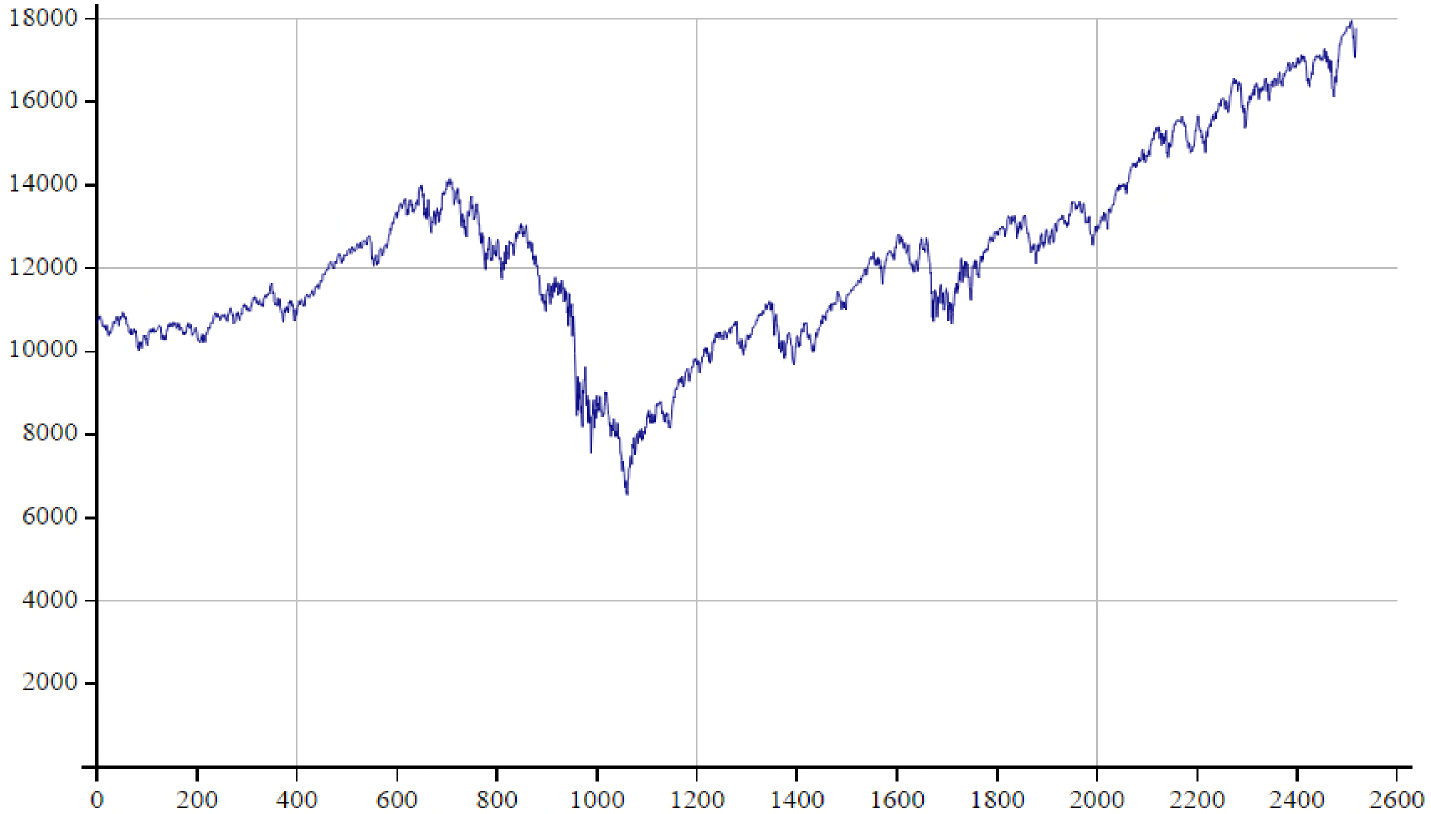
```
guess.m
M = 24;
totalcorrect = 0;
for k = 1:1000000
    correct = 0;
    term = randperm(M);
    for j = 1:M
        if term(j) == j
            correct = correct + 1;
        end
    end
    totalcorrect = totalcorrect + correct;
end
totalcorrect/1000000
```

Stock Market With APL: Looking and Predicting ****

```
Get data click→ http://jmb.aplcloud.com/jbgames/Data/DJIACleaned.txt
Then save this file as maybe→ DJIACleaned.txt somewhere on your computer
Top line of file has DATE VALUE the rest have the data. import needs this.
      D←import '' ⍝ from APL choose your downloaded file DJIACleaned.txt
```

```
⍝ Inspect the data(2004 to 2014) we read from file into namespace D:
      D.⎕NL 2                ⍝ shows all variables in namespace D
DATE
VALUE
      ⍴D.DATE                ⍝ show # of dates (⍴)
2609                         ⍝ 2609 dates(from 2004 to 2014)
      ⍴D.VALUE
2609                         ⍝ and 2609 stock values each of 2609 dates
      5↑¨D.DATE D.VALUE      ⍝ show 1st 5 dates and then stock values
20041220 20041221 20041222 20041223 20041224  10661.6 10759.43 10815.89
10827.12 0
      ↑5↑¨D.DATE D.VALUE     ⍝ ↑change nested vector to matrix to see better
20041220   20041221    20041222    20041223    20041224
   10661.6    10759.43    10815.89    10827.12          0
```

```
⍝ Clean data:
⍝ 1)keep only DATEs and VALUEs for VALUEs ≠ 0 (elim Sundays/Holidays)

      ⍴¨D.DATE D.VALUE←(⊂D.VALUE≠0)/¨D.DATE D.VALUE
2518 2518            ⍝ ⍴¨ shows 2,518 values left (down from original 2609)

      plotxy (⍳⍴D.VALUE)(D.VALUE) ⍝ try this to see quick plot 2518 days
```



```
⍝ Analyze the data 2004-2014
      (⊂D.VALUE=⌊/D.VALUE)/¨D.DATE D.VALUE
20090309  6547.05    ⍝ lowest stock market day was March 9, 2009

      (⊂D.VALUE=⌈/D.VALUE)/¨D.DATE D.VALUE
20141205  17958.79   ⍝ highest stock market day was Dec 5, 2014

⍝ lets get some day to day differences in stocks now

D.DIF← -2-/D.VALUE    ⍝  (-2-/) takes day to day differences & changes sign

5↑D.VALUE                ⍝ Show first 5 days of Dows
10661.6 10759.43 10815.89 10827.12 10776.13

4↑D.DIF                  ⍝ Show first 4 Dow differences(3 ups & 1 down)
97.83 56.46 11.23 ¯50.99

+/D.DIF>150              ⍝ how often Dow up > 150 points in one day
209                      ⍝ 209 days

+/0<(¯1⌽D.DIF>150)/D.DIF ⍝ how many times did it rise again the next day
100                      ⍝ 100 days (from total of 209 rise days)

avg (¯1⌽D.DIF>150)/D.DIF ⍝ average amount of change day after 150 pt rises
¯15.26492823             ⍝ ¯1⌽ rotates data by 1 so selects day after rise

+/D.DIF>0                ⍝ how many times did Dow go up at all in one day
1355                     ⍝ 1355 days(remember total days was 2609)

avg D.DIF>0              ⍝ average # days it rose at all
0.5383392928             ⍝ 1355/2518 equals about 54% (little more than ½)

avg D.DIF                                         ⍝ Average daily stock change.
2.827393723                                       ⍝ It rises avgerage <3 a day.

      (⊂0,D.DIF=⎕←⌊/D.DIF)/¨D.DATE D.VALUE   ⍝ when was the biggest fall
¯777.68                                           ⍝ 777.68 points lost
20080929 10365.45                                 ⍝ 09/29/2008 fell to 10365.45

      (⊂0,D.DIF=⎕←⌈/D.DIF)/¨D.DATE D.VALUE   ⍝ when was the biggest rise
936.42                                            ⍝ 936.42 points up
20081013 9387.61                                  ⍝ on 10/13/2008 up to 9387.61
```

```
⍝ But what if market drops 600+ pts? What should you do the next day?
avg (¯1⌽D.DIF<¯600)/D.DIF ⍝ average rise next day after down day
291.696 ⍝ so if market down buy next day if up sell next day. Try ¯400 or?
```

```
⍝ Your turn. Noodle around, learn APL and stock market! Happy Investing!
```

More Stock Market Calculations***

In this section we will play with the stock market some more to see which years, months, weeks and days might be best for stocks. First we need to break **D.DATE** up into **D.YR D.MONTH D.DAY** and **D.WKDAY**. This is done below by enclosing(⊂) # **D.DATE** which when formated(⍕) is an 8 long character string for each date. The first # 20041220 is broken into 3 chars using the 1's in the string 1 0 0 0 1 0 1 0 for YR MONTH DAY like this 2004 12 20. YMD is a vector of 2518 pieces. The 1st contains 2004 12 20, the 2nd 2004 12 21 etc.

(for example ⍎¨1 0 0 0 1 0 1 0⊂⍕20041220 would produce 3 #'s 2004 12 20
The execute(⍎) each(¨)converts the char strings(from ⍕ back to numbers))

```
D.(YR MONTH DAY)←↓⍉↑YMD←⍎¨¨(⊂1 0 0 0 1 0 1 0)⊂¨⍕¨D.DJ[;1] ⍝split each date
D.WKDAY←7|-38339-days ¨YMD ⍝ 7 days in week. 2004 12 20 is a Monday=1
⍝ note: 38339=day before 2004 12 20. days returns days since 1899-12-31
```

Now lets see which weekdays, months, years, and weeks of month were best.

```
     2⍕{avg(⍵=1↓D.WKDAY)/D.DIF}¨⍳5 ⍝ avg close each week day to 2 decimals
¯0.64 9.83 1.15 2.30 1.16 ⍝ lowest close=Mon & highest=Tues

     2⍕{avg(⍵=1↓D.MONTH)/D.DIF}¨⍳12 ⍝ so Best months 3 & 4, worst 1 & 6
¯5.59 2.39 9.30 13.53 ¯3.44 ¯8.87 8.69 ¯1.72 5.54 2.62 4.76 6.92

     2⍕{avg(⍵=1↓D.YR)/D.DIF}¨  2003+⍳11 ⍝ Best years 2004, 2013 worst 2008
15.18 ¯0.26 6.95 3.19 ¯17.74 6.55 4.56 2.54 3.55 13.78 4.92

In the next example month is divided into 4 approximately equal segments of
about 8 days(last segment will be  <8 depending on days in month).
      2⍕{avg(⍵=1↓⌈D.DAY÷8)/D.DIF}¨⍳4 ⍝ ÷ days 1-31 by 8 & round up(⌈)
 1.72 2.80 1.10 6.43 ⍝ result last week in month stocks go up much more
```

The Power of 11 ***

11 is an important number. It is used as a verification check for many things such as 10 digit book bar codes, overcoming skips or scratches on CDs and in all sorts of internet communications where static etc causes losses. By using 11 lost parts of information can be identified so all the data does not have to retransmitted.
Look at http://www.numberphile.com/ & click on 11-11-11 Eleven link.

In the book industry when 10 digit bar codes are used the 10 digits are always selected in a way so the check number is evenly divisible by 11. This is explained on the video link above. Here is an example:

Here is a barcode: 0 3 1 2 1 5 2 2 7 2 from book **Tongue-Fu** by Sam Horn.

Each of these numbers is multiplied by a number from 10 to 1.

```
 0  3  1  2  1  5  2  2  7  2 bar code
10  9  8  7  6  5  4  3  2  1 numbers from 10 to 1
-----------------------------
 0 27  8 14  6 25  8  6 14  2 resulting multiplication
```

The sum of (0 27 8 14 6 25 8 6 14 2) is 110 which is divisible by 11: (110÷11=10) . All 10 digit barcodes on backs of books when multiplied like this and added up are divisible by 11. This is called the checksum.

Here's how to do this in APL. First enter the program like this:

```
      barcode11←{0=11|+/⍵×⌽⍳10}
```

and test it like this:

```
      barcode11 0 3 1 2 1 5 2 2 7 2       ⍝ good bar code
1                                         ⍝ 1 means good, 0 would be bad
```

computer returns 1 for yes if it is divisible by 11. A bad barcode will result in a 0.

```
      barcode11 7 3 1 2 1 5 2 2 7 2        ⍝ bad barcode
0                                          ⍝ 0 means bad, 1 would be good
```

Here is how it works from right to left: The program **{0=11|+/⍵×⌽⍳10}** generates the numbers 1-10(⍳10), reverses them (⌽) and multiples the reversed numbers(10-1) by **⍵**(which is the barcode read into the program) then sums the resulting numbers up(+/) and finds the residue or remainder(|) of division by 11. If the residue equals(=) 0 that means the sum is evenly divisible by zero with nothing left over(no residue) and the program returns a 1(if true that 0=the residue) or 0(if 0≠ the residue)

Here is an example using residue(|):

```
       13|26 28 30   ⍝ remainder(|) of 13 divided into each # 26 28 30
0 2 4  ⍝ 13 into 26 has no remainder. 13 into 28 residue is 2 and 30 is 4
```

Now I was curious how good this barcode check was so I tested it by taking a valid(divisible by 11) bar code and randomly changing 1 number and checking the new number to see if would indeed fail the divide by 11 check.

I wanted to check it in a lot of ways to be certain this barcode method would catch all slight changes, so I wrote a program to randomly change one number in a 10 digit bar code. Here is my program:

```
      change1←{c[i]←((¯1+⍳10)~((i←?10)⊃c←⍵))[?9] ⋄ c}
```

Here's how it works. 1st there are 2 commands, diamond(⋄) separates them.

c[i]←((¯1+⍳10)~((i←?10)⊃c←⍵))[?9] This part determines a random number to insert into random i^{th} position(**c[i]←**) of my changed string c. First the changed string is created by copying the old string (**c←⍵**). Next, a random position to change(**i**) between 1-10 is made by (**i←?10**). The code:(**¯1+⍳10**) gets the numbers 1-10 and adds a negative 1(**¯1**) to each resulting in the numbers 0-9. The ~((**i←?10)⊃c**) part finds the value currently in position **i** of **c** and eliminates(~) it from the numbers 0-9 found by:(**¯1+⍳10**) so I am left with only the 9 new possible numbers to insert in **c[i]**. The **[?9]** part selects one of these 9 new numbers which is placed in (**c[i]←**).

c by itself after the diamond(⋄) simply tells the program to return the entire changed barcode(c) back to be displayed when the program is called:

```
      X←0 3 1 2 1 5 2 2 7 2 ⍝ for convenience store good barcode in X
      change1 X
0 3 1 2 6 5 2 2 7 2         ⍝ #1 random change 5th digit to 6
      change1 X
0 3 1 2 2 5 2 2 7 2         ⍝ #2 random change 5th digit to 2(same pos)
      change1 X
0 3 1 2 1 5 2 2 2 2         ⍝ #3 random change 9th digit to 2
      change1 X
0 3 1 2 6 5 2 2 7 2         ⍝ #4 random change 5th digit to 6(same as #1)
```

Now I can check these to see if they fail the divide by 11 check.

```
barcode11 0 3 1 2 6 5 2 2 7 2
0
```

The zero means it failed the check. Indeed all these 1 digit changes fail the check. This is promising but I need to do much more checking to be sure so I need to simplify things some more to get more efficient.

First I can put the two programs together to check more quickly like this:

```
      barcode11 change1 X
0
```

change1 changes 1 random # of barcode in X & then barcode11 checks that #

If I wanted to see the change & check it too I could do this.

```
      c,'check=',barcode11 c←change1 X
0 3 1 2 7 5 2 2 7 2 check=0       ⍝ X with 1 # changed(7) fails the check.
```

This shows the changed code and that it failed the check.

However, this is still not a very extensive check, so I did the following which does 100,000 random changes on the string(X) and adds up how many pass the check. The result was zero, meaning none of the changes pass the check, so I feel pretty confident that the 11 barcode check method is a good one. Here is the program that does the 100,000 check.

```
      +/barcode11¨change1¨ 100000⍴⊂X
0                            ⍝ none of the 100,000 new strings passes check
```

Here is how this works. First I made up 100,000 X strings with the same valid barcode. The enclose (⊂) symbol takes the 10 digit string(X) and puts in a packet and then **100000⍴** makes 100000 of these packets. The each operator (¨) tells the programs to operate on each of the 100,000 X string packets. The **change1** program grabs each(¨) of these same good string packets and makes one random change in each and passes it to the **barcode11** program which checks each(¨) of the 100,000 new string packets and returns a string of 100,000 0's and 1's indicating if each changed string passed the divide by 11 check. Finally the string of 100,000 0's and 1's is added up (**+/**) and the result is zero which is displayed and tells us none of the 100,000 random changes was valid.

Mortgage Calculations ****

Sample: from Wikipedia http://en.wikipedia.org/wiki/Amortization_schedule

Problem: You want to buy a $100,000 apartment in Waikiki. Should you get a loan for 7% for 20 years or 4% for 30 years? Two things are relevant here. 1) Which loan has lower monthly payment? 2)What is total cost of each loan?

P=Principle i=monthly interest .07÷12months n=#payments:20yrs×12months

```
    P←100000 ⋄ i7 i4←.07 .04÷12 ⋄ n20 n30←20 30×12 ⍝ assign values
    MonthlyPaymentAnuityFormula←{P i n←⍵ ⋄ P×i+i÷((1+i)*n)-1} ⍝ define
    PresValOfAnuity←{A i n←⍵ ⋄ (A÷i)×1-1÷(1+i)*n} ⍝ define
```

Explore: Monthly payment for each loan(MP720 and MP430).

```
    +MP720 MP430←MonthlyPaymentAnuityFormula¨(P i7 n20)(P i4 n30)
```

```
775.2989356 477.4152955 ⍝ So monthly is much less for 4% 30 year loan.
```

Explore: How loan and interest payments change over time in these loans

```
    PresValOfAnuity¨ (MP720 i7 (n20-7×12))(MP430 i4 (n30-7×12))
79267.91062 86059.4709 ⍝ MP430 owes more after 7 years of payments
```

```
    P×i7 i4             ⍝ Initial interest paid(Principle×interest rate)
583.3333333 333.3333333 ⍝ Starting interest payment higher for 7% rate
```

```
    MP720 MP430-P×i7 i4 ⍝ Initial pay to Prin (monthly paym-interest paid)
191.9656023 144.0819621 ⍝ So initially 7% loan is paying off quicker
```

```
    (P-191.97 144.08)×i7 i4 ⍝ 2nd interest payment (on prin-prev prin pay)
582.2135083 332.8530667 ⍝ each pay less as part of loan is paid each month
```

```
    MP720 MP430-582.21 332.85      ⍝ 2nd payment to Principle
193.0889356 144.5652955 ⍝ < interest paid so more to principle
```

create fns:)ed amort, press enter, type lines below in edit window, when done press ESC & fns created & you back in session ready to try the fns.

```
  amort←{P i n←ω ⍝ monthly payment table= Principle, Interest & Balance
     mp←{P i n←ω ⋄ P×i+i÷((1+i)*n)-1}P i n        ⍝ fns mp=monthly payment
     pval←{A i n←ω ⋄ (A÷i)×1-1÷(1+i)∘.*⌽⍳n} ⍝ fns pres value every payment
     int←i×bal←pval mp i n ⋄ prin←mp-int ⋄ bal←bal-prin  ⍝ get all results
     lbl←'Period' 'PrinPay' ' IntPay' '  Balance'     ⍝ make column labels
     tbl←lbl⍪(⍕¨⍳n),(2⍕¨prin),(2⍕¨int),[1.5](2⍕¨bal)     ⍝ put all together
     tbl⍪(⊂'Total Paid'),(2⍕¨+/¨prin int),⊂2⍕0  ⍝ sum principle & interest
 }
```

Answer: call **amort** & get result. Last row is answer for cost of 7% loan.

```
      amort P i7 n20  ⍝ call fns: loan payback table 240 rows(20×12).
Period       PrinPay   IntPay   Balance
1            191.97     583.33    99808.03
2            193.09     582.21    99614.95
.............................................. ⍝ rows deleted for brevity
239          766.33     8.97      770.80
240          770.80     4.50      0.00 ⍝ final payment and bal=0
Total Paid   100000.00  86071.74  0.00 ⍝ principle + $86071 interest Ouch!
```

Now try: **amort P i4 n30**. Is 4% cheaper than above $86,071.74 for the 7% loan? Please note that though it is stated to be a 7% loan it is 7% every year & becomes 86% in 30 years. Borrowing is expensive. Become a Banker!

Roots of a Polynomial ****

Given an equation such as $y=2x^2 +1x -10$. what are it’s roots(the x values that cause y to be equal to 0). Here is an APL program to find them:

```
      quadsim←{a b c←ω ⋄ d←(b*2)-4×a×c ⋄ (+/x),-/x←((-b),d*.5)÷2×a}
```

```
      quadsim 2 1 ¯10           ⍝ try program equation: y=2x^2 +x -10
2 ¯2.5                          ⍝ so if x=2 or ¯2.5 the equation for y = 0
```

to check the result: substitute 2 and ¯2.5 into the equation

```
     x←2 ¯2.5                      ⍝ store roots in x
     (2×x*2)+x+¯10                 ⍝ test the equation with values of x.
0 0                                ⍝ 0 0 result so 2 & ¯2.5 are roots of eq.
```

The above check is clear but there is an even easier way in APL.

APL has a special symbol to insert values into equations of this general type. It will also work for higher order equations like $3x^5+2x^3+x^2+5$. For this equation if x←⍳6 then (x⊥¨⊂3 0 2 1 5) would result in the numbers 1-6 being inserted in the equation $3x^5+2x^3+x^2+5$ resulting in: 11 63 269 809 1935 3971. This makes it very easy to make y values from the x values or to test to be sure the roots found are correct(result=0).

```
      2 ¯2.5⊥¨⊂2 1 ¯10           ⍝ test x=2 ¯2.5 as roots of 2x$^2$ +x -10
0 0                              ⍝ 0 0 result so 2 & ¯2.5=roots of: 2x$^2$ +x -10
```

But not all equations have 2 roots, some equations have only one root and others have only imaginary roots. Here are two APL program to calculate any of these possible cases the first labels the result the second just returns the roots which can then be passed on to other APL programs. How many roots there are can be determined by the sign(×) of the calculation of **disc**. If sign of disc=1(positive) there are two real roots, if sign of disc=0(zero) there is one real root and if sign of disc=¯1 there are two imaginary roots. Here is the complete program with labeled output for the 3 cases:

```
QUAD←{⍝ roots of equation e.g.  QUAD 2 1 ¯10  for: 2×x*2 +1×x -10
     a b c←⍵ ⋄ d←(b*2)-4×a×c
     d>0:'2 Real Roots:',(-b+1 ¯1×d*0.5)÷2×a
     d=0:'1 Real Root',-b÷2×a
     d<0:'2 Complex Roots',(u,'+',v,'I'),' and',((u←-b÷2×a),'-',(v←((-
d)*0.5)÷2×a),'I')
 }
```

To create this fns type **)ed QUAD** press enter & type lines into editor. Lets test it out with the same example then with 2 other equations:

```
      QUAD 2 1 ¯10  ⍝              ⍝ 2x$^2$ +x -10
2 Real Roots: ¯2.5 2
      QUAD 3 ¯2 10                 ⍝ 3x$^2$ -2x +10
2 Complex Roots 0.333333 + 1.795054 I and 0.333333 - 1.795054 I
      QUAD 9 12 4                  ⍝ 9x$^2$ +12x +4
1 Real Root ¯0.6666666667
```

Here is a modified version of **quadsim** that returns only real roots unlabeled. This will be more useful to pass to plotting programs:

```
quad←{
     a b c←⍵ ⋄ d←(b*2)-4×a×c         ⍝ input ⍵ to a b c ⋄ find disc d
     d>0(-b+1 ¯1×d*0.5)÷2×a          ⍝ if disc>0 show 2 roots
     d=0:-b÷2×a                      ⍝ if disc=0 show 1 root
     d<0:⍬                           ⍝ if disc<0 show nothing(⍬)
 }
```

Lets try same 3 equations at once. Note: **display** is APL fns to display results so you can see their structure. **display** is used for display only, not when passing results to other programs.

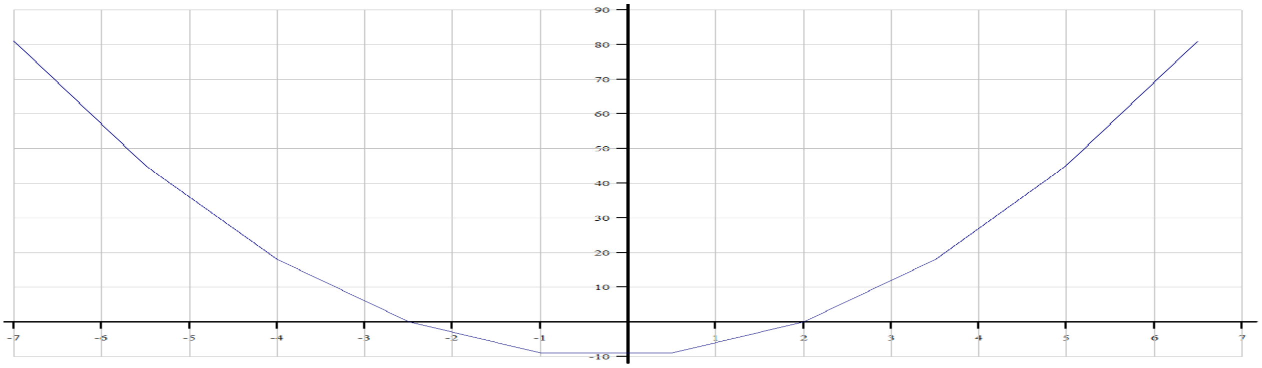
```
      display quad ¨(2 1 ¯10)(3 ¯2 10)(9 12 4)
┌→───────────────────────────┐ ⍝
│¯2.5 2│0│¯0.6666666667│ ⍝ 2 roots, no roots, 1 root
└~─────→┴⊖┴~────────────┘
```

Now lets plot equation 2x^2 +x -10 so we can see its shape and where the roots are. First we need to generate some x plotting values around the roots of ¯2.5 and 2 so we can see these critical points clearly in the upcoming plot. The program **xaroundroots** below does that. It takes the two roots as input on the right and the number of x values to make on the left. It then finds the difference(**dif**) between the two roots and generates **α**(50) x values from the lower root(**d**) minus the difference to the upper root(**u**) plus the difference so in this case the difference between roots ¯2.5 and 2 is 4.5 so 4.5 is subtracted from ¯2.5 giving ¯7 which is the first x value as can be seen below. Then it takes the upper root which is 2 and adds 4.5 to that giving 6.5 which is the highest of the 10(α) x values. If only 1 root it makes a guess at what would be a reasonable range.

```
xaroundroots←{α←50 ⍝ find α # of values around roots
     u d dif←{ ⍝ nested dfns to upper lower and diff
         2=⍴,⍵:u,d,((u←⌈/⍵)-d←⌊/⍵)           ⍝ dif if 2 roots
         1=⍴,⍵:u,d,((u←⍵+5⌈|⍵÷2)-d←⍵-5⌈|⍵÷2) ⍝ dif if 1 root
     }⍵  ⍝ if 1 root near 0 sets to range of about 30
     du←(d,u)+(-dif),dif
     (1⊃du)+(¯1+⍳α)×(-/⌽du)÷α-1
 } ⍝ make α x values in range(1⊃du to 2⊃du)
```

To create this fns type: **)ed xaroundroots** then enter & type above lines

```
      2⍕X←10 xaroundroots ⎕←quad 2 1 ¯10 ⍝ show roots & make 10 X values
¯2.5 2
 ¯7.00 ¯5.50 ¯4.00 ¯2.50 ¯1.00 0.50 2.00 3.50 5.00 6.50

      2⍕Y←(2×X*2)+X+¯10 ⍝ put above X values into equation to get Y’s
 81.00 45.00 18.00 0.00 ¯9.00 ¯9.00 0.00 18.00 45.00 81.00

      2⍕DATA←Y,[.5]X ⍝ put the X and Y values into a matrix for plotting.
 81.00 45.00 18.00  0.00 ¯9.00 ¯9.00 0.00 18.00 45.00 81.00
 ¯7.00 ¯5.50 ¯4.00 ¯2.50 ¯1.00  0.50 2.00  3.50  5.00  6.50
      plotxy X Y                        ⍝ Now Plot the 10 points
```



So now lets put this together in a little program so we can do it easily:

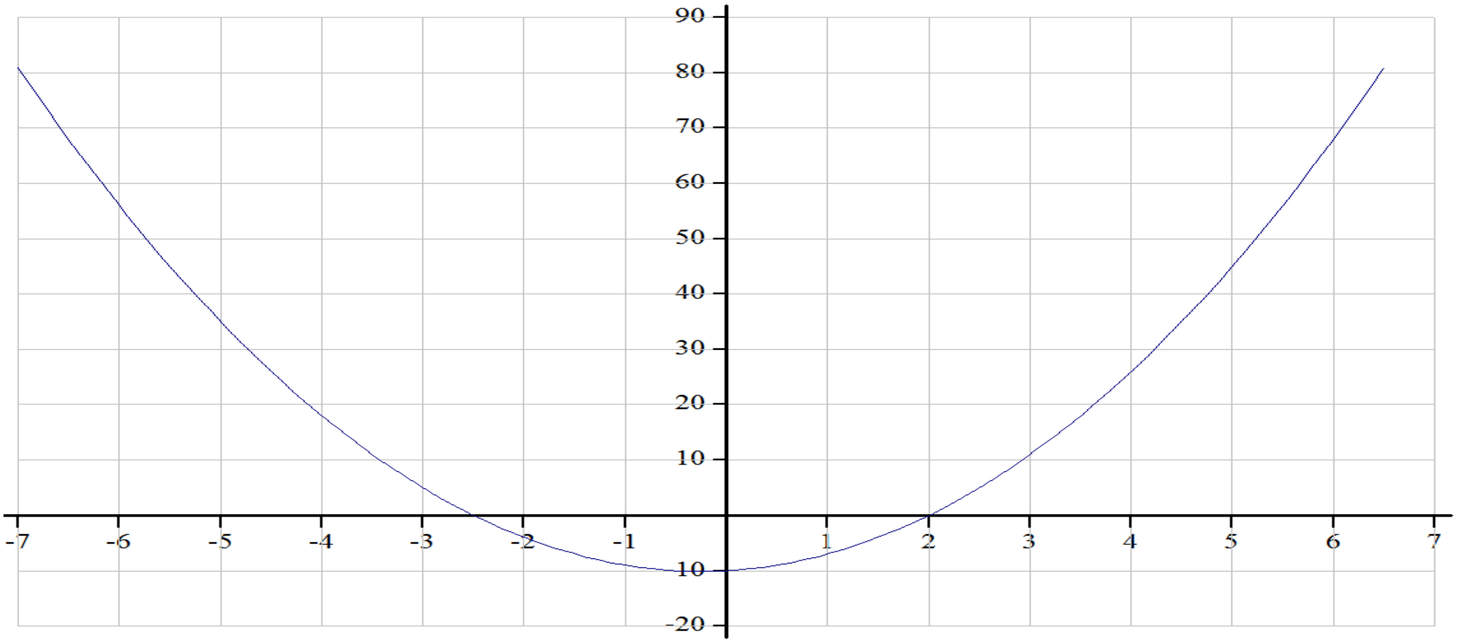
```
   rootsandplot←{α←100 ⋄ ch.Set'Footer' ⎕←ft←quad ⍵
                 x←α xaroundroots ft ⋄ y←x⊥¨⊂⍵ ⋄ plotxy x y}
```

Notes: this program really has 4 lines separated by diamonds (⋄)
1. α←100 sets default to make 100 x & y values. If you don't specify a number on the left when you call the program you will get 100 x & y values.
2. ⎕← displays roots computed by quad program using input equation (⍵) and then passes the roots to **xaroundroots** which finds 100 x values near the roots so we will have a good plot around the roots.
3. **y←x⊥¨⊂⍵** puts the found x values into the equation(i.e. ⍵=2 1 ¯10). As mentioned above this tricky code is APL equivalent to y←(2×x*2)+x+¯10
4. finally **plotxy** passes **x** and **y** to little plot program I wrote to make a pretty display. Here it is:

```
  R←{ax0}plotxy data
   ⍝ plot data:x=col1 y=col2 or x=vector1 y=vector2
   ax0←0=⎕NC'ax0' ⍝ if no ax0 axes cross at 0
   :If 2=≡data ⋄ data←⍉↑data ⋄ :End
   ch.Set'Lines' 1 2 4 5
   ch.Set¨(ax0,ax0,1)/('Xint' 0)('Yint' 0)('XYPLOT,GRID')
   ch.Plot data ⋄ PG←ch.Close
   R←'View PG ⍝ to see it'
```

To create this fns type **)ed plotxy** press enter and type lines into editor.
And enter line **)copy rainpro** to bring in the fancy APL graphics.
Now lets the try program **rootsandplot** for the equation: $2x^2$ +x -10

```
      rootsandplot 2 1 ¯10 ⍝ call program shows roots and plots xy data
¯2.5 2                      ⍝ shows roots. Plots 100 xy value pairs
View PG ⍝ to see it      ⍝ just press enter on this line to see plot
```

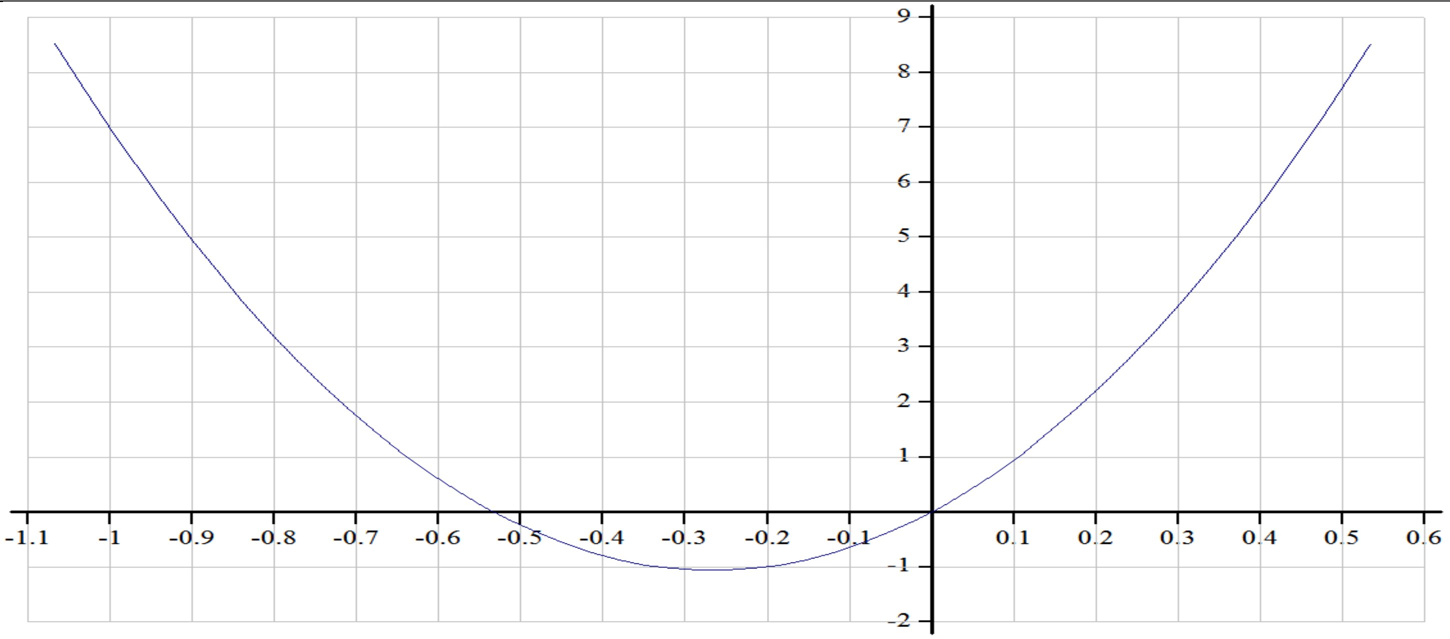


Notice where the roots(y=0) ¯2.5 and 2 are on the plot.

This program will plot data for any polynomial with 2 real roots simply by entering the parameters for x^2 x^1 and x^0.

Now lets try: $y=15x^2 + 8x + 0$ and request only 50 values to plot.

```
      50 rootsandplot 15 8 0     ⍝ plot 50 xy points for y=15x^2 + 8x + 0
¯0.5333333333 0                  ⍝ shows 2 roots.
View PG ⍝ to see it     ⍝ just press enter on this line to see plot
```

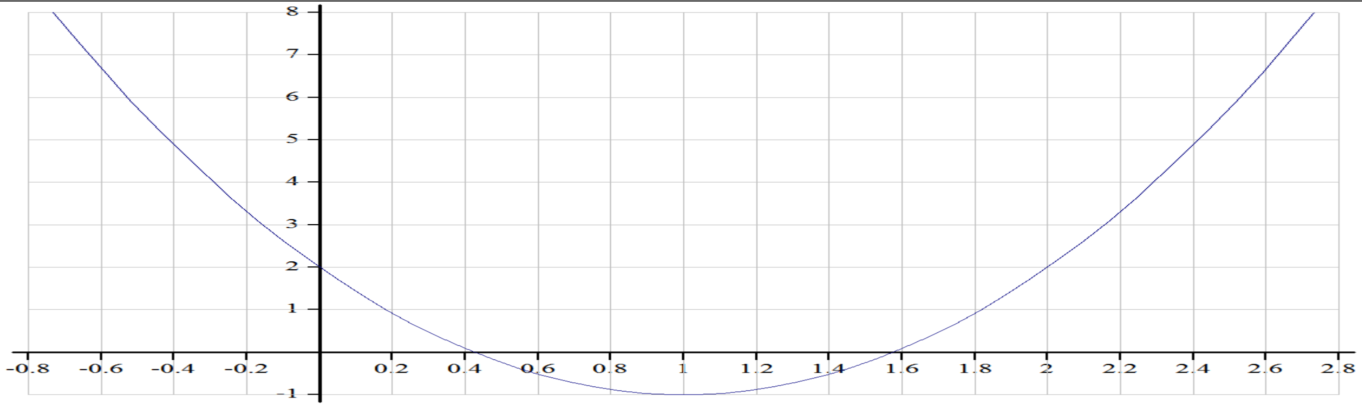


Again notice where roots are and see that xaroundroots centers plot nicely

Quadric Equations and Functions *****

Quadric equations are of the general form y=ax*2 + bx +c . The above program only works when there are two roots and it does not tell us either vertices or minimums of the function. So here is a more complete program. Lets plot such an equation in APL. Lets try y=3x*2 + ¯6 + 2 . Here is a way to plot such an equation in APL by entering just a b and c into the program.

```
      QuadPlot 3 ¯6 2
View PG ⍝ to see it
```



y=ax*2+bx+c a= 3 b= ¯6 c= 2 Function min at x y= 1.000 ¯1.000 xintercepts= 0.423 1.577

The program footnote tells us that the function has a minimum at x=1 y=¯1 and that it crosses the x axis twice, once at .423 and again at 1.577. Here is the QuadPlot program:

To create this fns type **)ed QuadPlot** press enter & type lines into editor.

```
QuadPlot(a b c);x;y;xint;xvert;yvert;mm;rng;ft
 ⍝ Plot quadratic eq: QuadPlot 2 ¯1 ¯7  for: 2x*2 -x -7 (a=2 b=¯1 c=¯7)
 mm←((1+a<0)⊃'F min' 'F max'),' at x y='  ⍝ "max" if a<0 otherwise "min"
 xvert←-b÷2×a                             ⍝ xvert=where y is min or max
 yvert←xvert⊥¨⊂a b c                      ⍝ solve eq for yvert using xvert
 xint←,quad a b c                         ⍝ quad formula for x intercepts
 rng←⍎(1+⍴xint)⊃'0,xvert' '0,xint' 'xint' ⍝ find range of x values to plot
 :If rng≡0 0 ⋄ rng←¯10 10 ⋄ :End          ⍝ fix range if at 0 0
 x←xaroundroots rng                       ⍝ find good x values to plot
 y←x⊥¨⊂a b c                              ⍝ solve eq for y using x values
 ft←'y=ax*2+bx+c  a=' 'b=' 'c='mm'xintcepts=',¨a b c(3⍕xvert
yvert),⊂3⍕xint
 ch.Set'Footer' ft
 plotxy x y
```

Integration: Find Area Below Any Equation in 1 Line APL ***

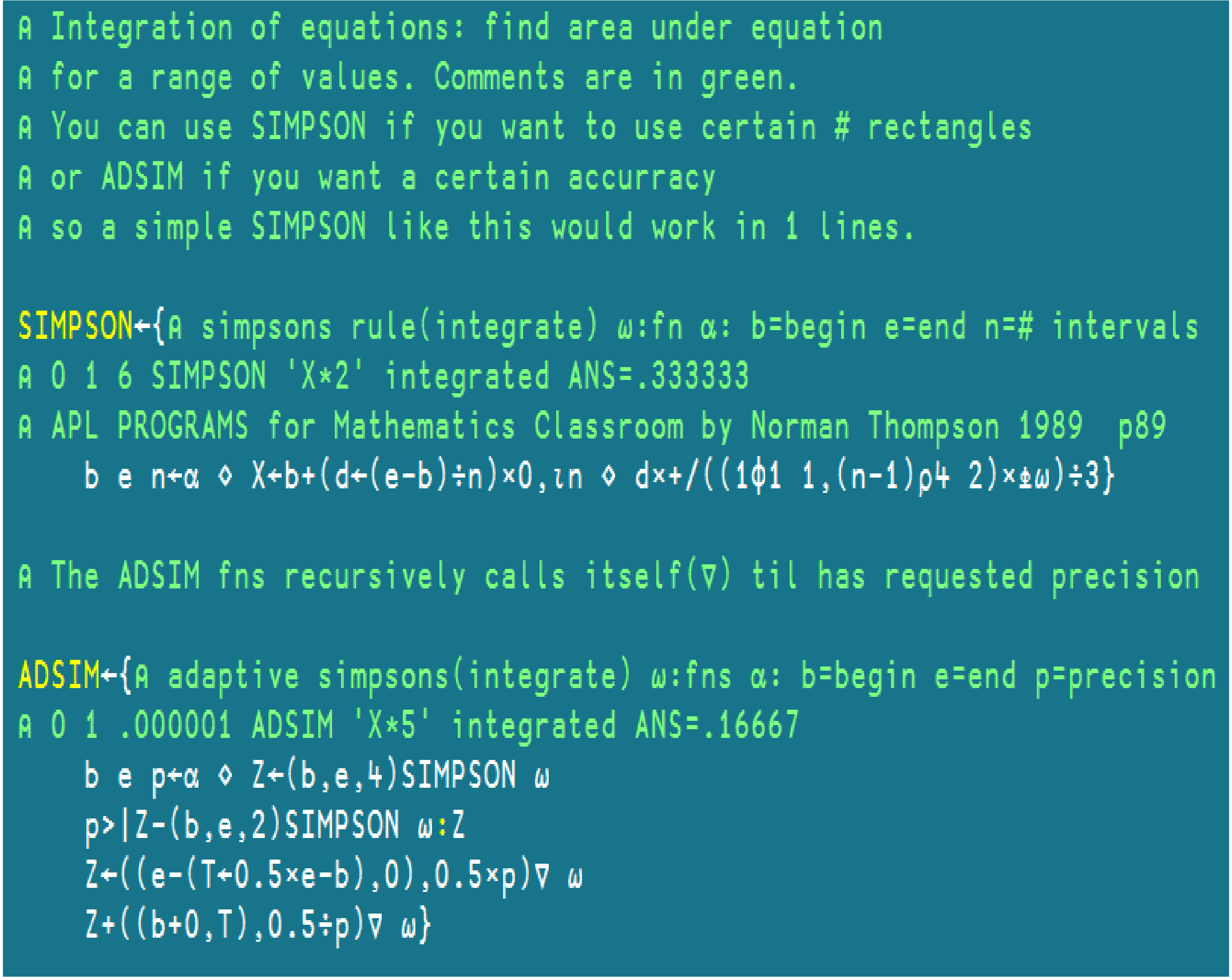
Define the fns:

```
SIMPSON←{b e n←α ⋄ X←b+(d←(e-b)÷n)×0,ιn ⋄ d×+/((1⌽1 1,(n-1)⍴4 2)×⍎ω)÷3}
```

Call fns: 0=begin interval 1=end interval 6=# rectangles X*2 is equation to integrate(ω).

```
      0 1 6 SIMPSON 'X*2'
0.3333333333
```

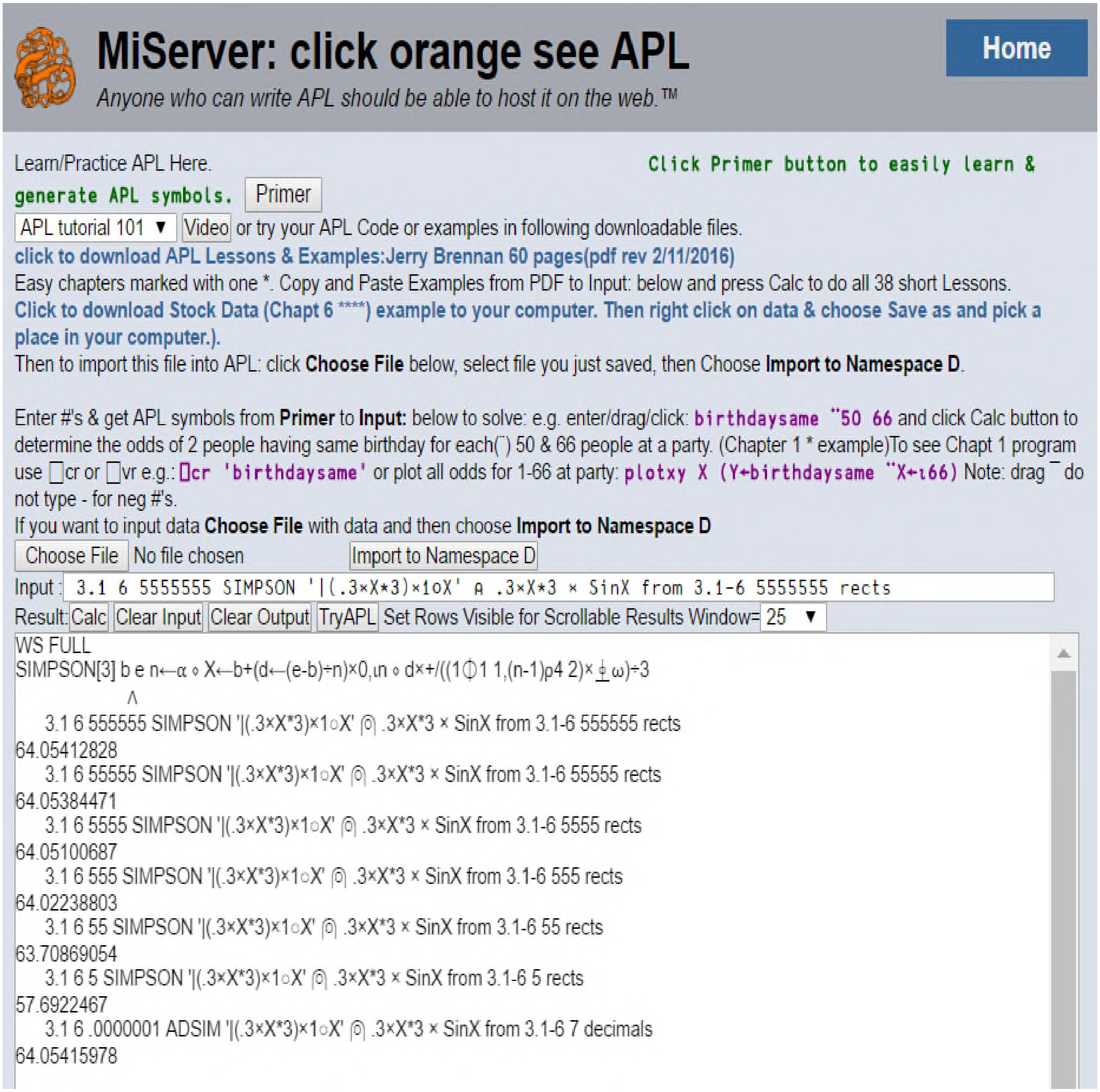
Here are the 2 actual fns with extensive comments in green. Actual code is only the white. You can use SIMPSON and tell fns how many rectangles you want or ADSIM if you want to set accuracy of result & the fns will figure out how many rectangles are needed for the desired level of accuracy.



```
⍝ Integration of equations: find area under equation
⍝ for a range of values. Comments are in green.
⍝ You can use SIMPSON if you want to use certain # rectangles
⍝ or ADSIM if you want a certain accurracy
⍝ so a simple SIMPSON like this would work in 1 lines.

SIMPSON←{⍝ simpsons rule(integrate) ω:fn α: b=begin e=end n=# intervals
⍝ 0 1 6 SIMPSON 'X*2' integrated ANS=.333333
⍝ APL PROGRAMS for Mathematics Classroom by Norman Thompson 1989  p89
    b e n←α ⋄ X←b+(d←(e-b)÷n)×0,ιn ⋄ d×+/((1⌽1 1,(n-1)⍴4 2)×⍎ω)÷3}

⍝ The ADSIM fns recursively calls itself(∇) til has requested precision

ADSIM←{⍝ adaptive simpsons(integrate) ω:fns α: b=begin e=end p=precision
⍝ 0 1 .000001 ADSIM 'X*5' integrated ANS=.16667
    b e p←α ⋄ Z←(b,e,4)SIMPSON ω
    p>|Z-(b,e,2)SIMPSON ω:Z
    Z←((e-(T←0.5×e-b),0),0.5×p)∇ ω
    Z+((b+0,T),0.5÷p)∇ ω}
```

Below is online version in action. Go to **jmb.aplcloud.com** & choose IntroLive button. To Practice using live APL paste line below to Input:

```
3.1 6 5555 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 5555 rects
```



MiServer: click orange see APL
Anyone who can write APL should be able to host it on the web.™
Home

Learn/Practice APL Here. Click Primer button to easily learn & generate APL symbols. Primer

APL tutorial 101 ▼ Video or try your APL Code or examples in following downloadable files.

click to download APL Lessons & Examples:Jerry Brennan 60 pages(pdf rev 2/11/2016)

Easy chapters marked with one *. Copy and Paste Examples from PDF to Input: below and press Calc to do all 38 short Lessons.

Click to download Stock Data (Chapt 6 ****) example to your computer. Then right click on data & choose Save as and pick a place in your computer.).

Then to import this file into APL: click **Choose File** below, select file you just saved, then Choose **Import to Namespace D**.

Enter #'s & get APL symbols from **Primer** to **Input:** below to solve: e.g. enter/drag/click: `birthdaysame ¨50 66` and click Calc button to determine the odds of 2 people having same birthday for each(¨) 50 & 66 people at a party. (Chapter 1 * example)To see Chapt 1 program use ⎕cr or ⎕vr e.g.: `⎕cr 'birthdaysame'` or plot all odds for 1-66 at party: `plotxy X (Y←birthdaysame ¨X←⍳66)` Note: drag ¯ do not type - for neg #'s.

If you want to input data **Choose File** with data and then choose **Import to Namespace D**

Choose File No file chosen Import to Namespace D

Input: `3.1 6 5555555 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 5555555 rects`

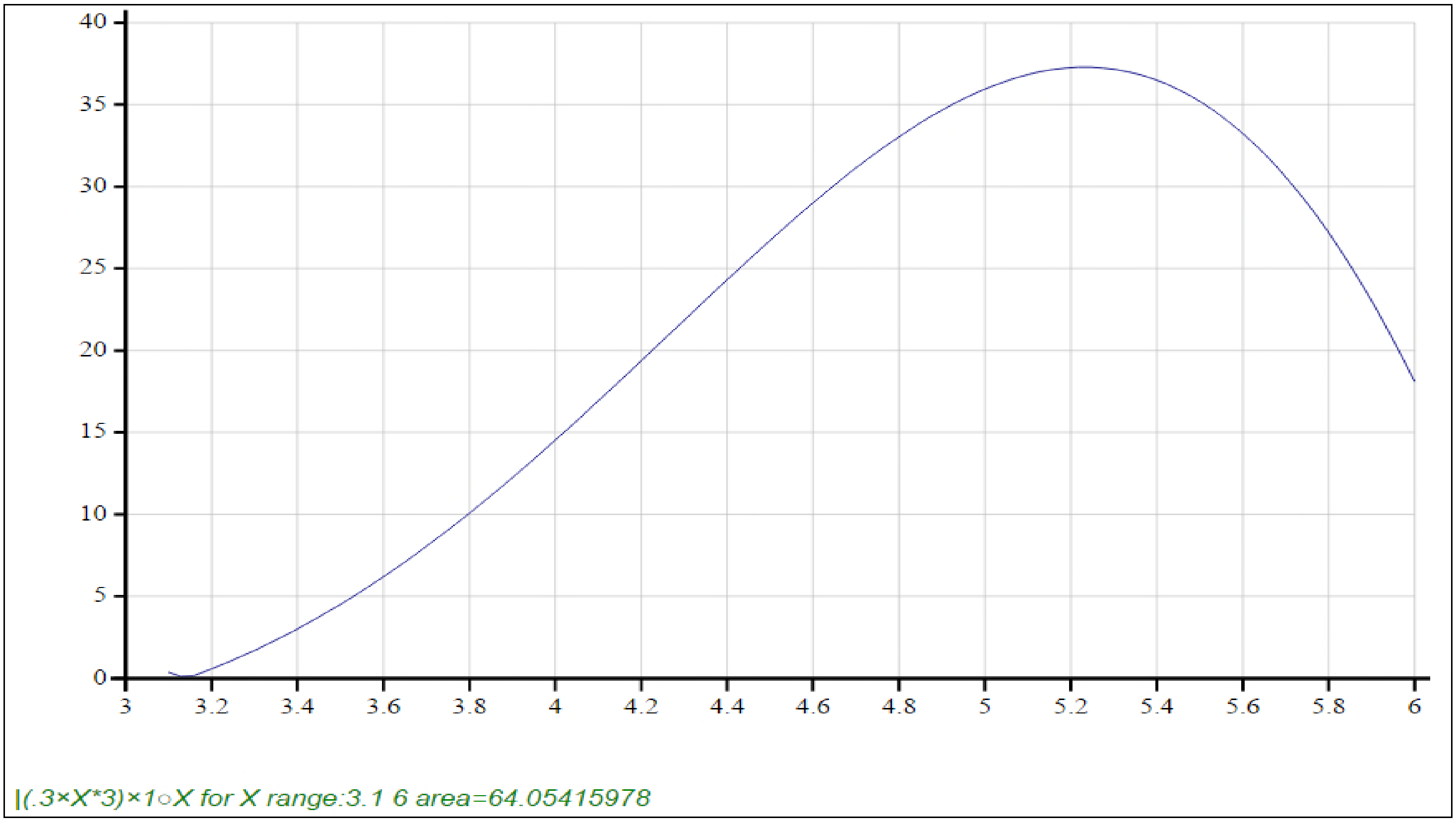
Result: Calc Clear Input Clear Output TryAPL Set Rows Visible for Scrollable Results Window= 25 ▼

```
WS FULL
SIMPSON[3] b e n←α ⋄ X←b+(d←(e-b)÷n)×0,⍳n ⋄ d×+/((1⌽1 1,(n-1)⍴4 2)×⍎ω)÷3
                ∧
      3.1 6 555555 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 555555 rects
64.05412828
      3.1 6 55555 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 55555 rects
64.05384471
      3.1 6 5555 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 5555 rects
64.05100687
      3.1 6 555 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 555 rects
64.02238803
      3.1 6 55 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 55 rects
63.70869054
      3.1 6 5 SIMPSON '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 5 rects
57.6922467
      3.1 6 .0000001 ADSIM '|(.3×X*3)×1○X' ⍝ .3×X*3 × SinX from 3.1-6 7 decimals
64.05415978
```

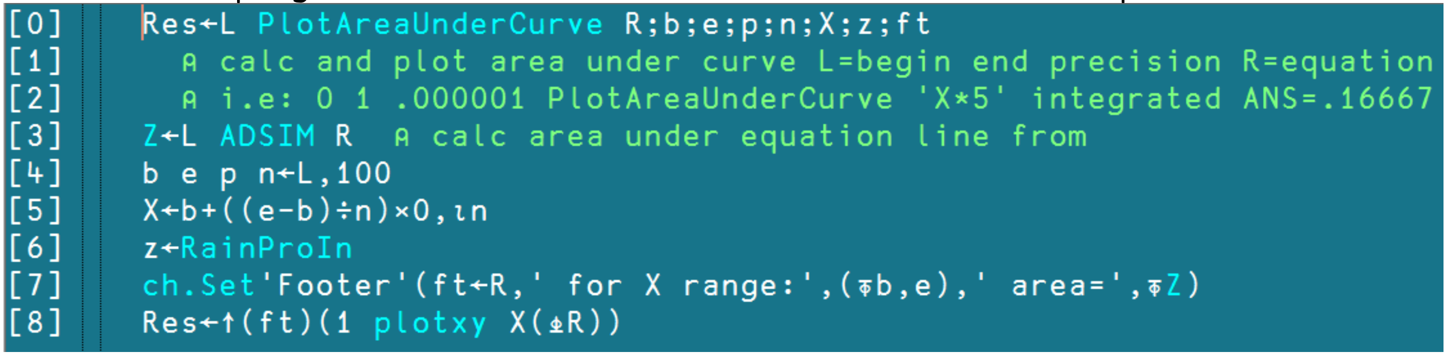
As you can see the correct answer to 7 decimal places is 64.05415978 as found by ADSIM. It requires a lot of very small rectangles added together to be that accurate. SIMPSON takes more than 555555 such rectangles to be accurate to only 4 decimal places and it overfills the workspace if I try for more. The much more efficient ADSIM however can easily find the answer to 7 decimal places.

To see a plot of the area cut/paste line below to Input: field on web page and press Calc below it.

```
3.1 6 .000001   PlotAreaUnderCurve '|(.3×X*3)×1○X'
```



Here is the program **PlotAreaUnderCurve** that calls **ADSIM** & plots the curve:



```
    Res←L PlotAreaUnderCurve R;b;e;p;n;X;z;ft
      ⍝ calc and plot area under curve L=begin end precision R=equation
      ⍝ i.e: 0 1 .000001 PlotAreaUnderCurve 'X*5' integrated ANS=.16667
    Z←L ADSIM R  ⍝ calc area under equation line from
    b e p n←L,100
    X←b+((e-b)÷n)×0,⍳n
    z←RainProIn
    ch.Set'Footer'(ft←R,' for X range:',(⍕b,e),' area=',⍕Z)
    Res←↑(ft)(1 plotxy X(⍎R))
```

The above problem is interactively discussed in excellent detail at http://www.intmath.com/blog/mathematics/riemann-sums-4715.

Email me at jbrennan@hawaii.rr.com if you want to learn more. OR
1)go to jmb.aplcloud.com
2)press IntroLive button
3)choose menu choice called:

Practice using live APL *All APL Examples from PDF available here to try*

Basic Statistics ***

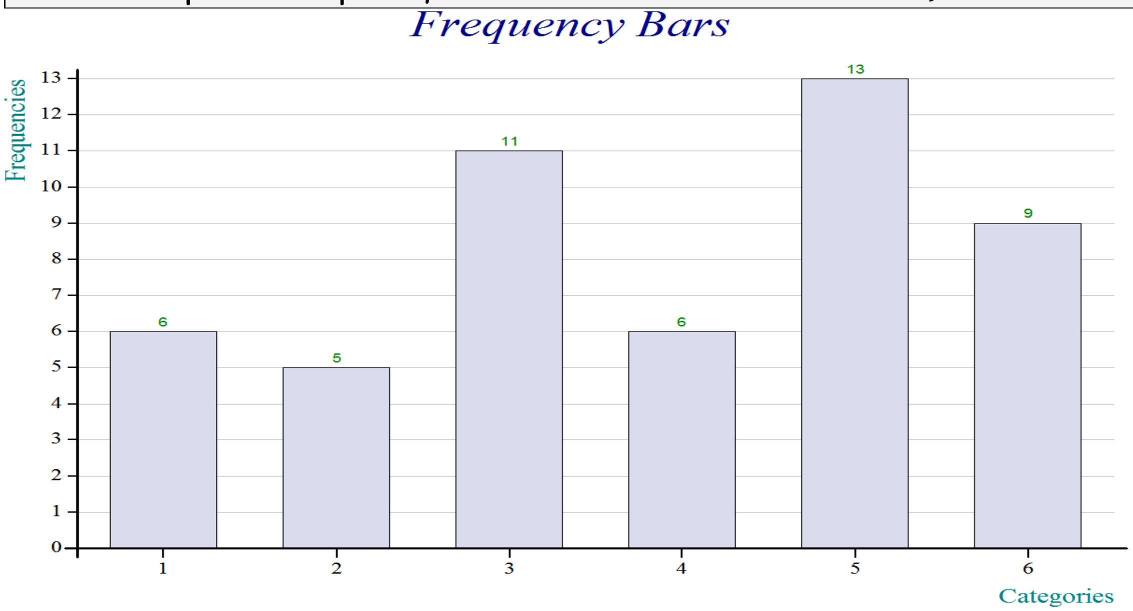
```
      Mean←{(+/⍵)÷(⍴⍵)}               ⍝ sum(+/) of #'s divided by #(⍴) of #'s
      Max←{⌈/⍵}                       ⍝ maximum of numbers(⌈/)
      Min←{⌊/ω}                        ⍝ minimum of numbers(⌊/)
      Range←{(max ω)-(min ω)}          ⍝ range - max minus min of numbers
      Sort←{ω[⍋ω]}                     ⍝ sort numbers up(⍋). ⍒ would sort down
      Median←{mid←(1+⍴s←sort ω)÷2 ⋄ Mean s[(⌊mid)(⌈mid)]}
      Variance←{(+/(ω-avg ω)*2)÷(¯1+⍴ω)} ⍝ sample variance
      Sdev←{(variance ω)*0.5}          ⍝ sample standard deviation
      Skew←{(+/(ω-Mean ω)*3)÷(¯1+⍴ω)×(Sdev ω)*3} ⍝ skew (+=right -=left)
      Kurtosis←{(+/(ω-Mean ω)*4)÷(¯1+⍴ω)×(Sdev ω)*4} ⍝ flatness -+ normal
```

The median is defined as the middle number if there are an odd # of #'s. If there are an even # of numbers its the average of the two middle numbers. The above median fns first sorts the data into s and finds the midpoint which is either a whole number position for odd # of #'s or a position 1/2 way between the two middle positions(mid) if there are an even number of numbers. After the diamond(⋄) the fns averages the 1 or two middle numbers s[(⌈mid)(⌊mid)]. Lets try a couple to see how it works.

```
      Median 5 6 8 7                    ⍝ s=5 6 7 8, mid=2.5, ⌊mid=2, ⌈mid=3
6.5                                     ⍝ s[2 3]=6 7, average of 6 7=6.5
      Median 9 5 8                      ⍝ s=5 8 9, mid=2, ⌊mid=2, ⌈mid=2
8                                       ⍝ s[2 2]=8 8, average of 8 8=8
      Freq←{↑(⍕¨u)(+⌿ω∘.=u←sort ∪ω)}  ⍝ define a frequency count fns Freq
      Freq num←?50⍴6            ⍝ call Freq with 50 rand(?) #’s(1-6) in num
 1   2  3  4   5  6             ⍝ here are the label #'s in row 1 of freqs
 9  10  7  7  11  6             ⍝ here are the frequencies in row 2 of freqs
```

The Freq function: u is sorted unique(∪)values of the 50 rand #'s(**NUM**) #'s are counted into unique categories 1-6 (+⌿ω∘.=u) & labeled(⍕¨u)

```
      FreqBar Freq ?50⍴6  ⍝ make 50 rand #’s 1-6, turn into freqs, plot
```



```
      Mode←{↑(⊂(f>1⌽f)∧(f>¯1⌽f))/¨v f←0,¨(↓Freq ω),¨0} ⍝ Mode uses Freq
      Mode NUM  ⍝ call mode program with same num used above for freq.
 2   6   ⍝ two modes one at 2 one at 6
11   8   ⍝ mode at 2 has a frequency of 11. mode at 6 has frequency of 8.
```

Modes are defined as any frequencies that are higher than frequencies immediately before or after it or are at either end and are higher than the one frequency that is either before it or after it. The program finds the frequencies. in this case for values 1-6:9 10 7 7 11 6 and puts 0's before and after the frequencies then compares by rotating(⌽) f to values before and(^) after and if it greater than both it is a mode as can be seen here:

```
      ↑v f (1⌽f) (¯1⌽f)   ⍝ This displays v f 1⌽f & ¯1⌽f as a 4 row table
0  1  2  3  4  5  6 0     ⍝ v are the values with 0's on each end
0  9 10  7  7 11  6 0     ⍝ f 10 & 11 only ones greater than(>) 1⌽f & ¯1⌽f
9 10  7  7 11  6  0 0     ⍝ 1⌽f 7 & 6 are less that 10 and 11 above them
0  0  9 10  7  7 11 6     ⍝ ¯1⌽f 9 & 7 are less that 10 and 11 above them
```

Kendall’s Tau : Rank Order Correlation ****

Here is some code for first example and then another example I found online. I also computed z score for it.
First your data in a1 and a2, then call the Ktau program. If two raters rated 8 bands numbered 1-8. Ktau computes how similar the rank orders are by counting concordances and discordances.

First put the bands in order by the ranks of the first rater a1. So a1 goes 1-8. Rater a2 had a different ordering. The both agreed in band 1. but rater a2 saw a1’s second best band as his 3rd best and a2 saw the 6th band as his second best. Now we determines concordances and discordances.

```
      a1←1 2 3 4 5 6 7 8
      a2←1 3 4 5 2 6 7 8
    ⍝  c=7 5 4 3 3 2 1 0  so c=25
    ⍝  d=0 1 1 1 0 0 0 0  so d= 3
```

So looking at the **a2** numbers band 1 had 7 concordances(7 numbers after it that were higher and 0 discordances(0 numbers after it that were lower). For **a2** band 2 had 5 concordances(5 numbers after it that higher) and 1 discordance(1 number after it that lower). Continuing for the other bands and adding them all we get 25 concordances and 4 discordances. The **Ktau** formula uses c and d like this. **Ktau=(c-d)÷(c+d)**. The **Ktau** fns below does this using a sub fns call **cd** in **[1]** which calls itself(using **∇**) repeatedly for each rank counting the numbers below that rank that are concordant or discordant Both **c** and **d** are calculated by fns **cd** in **[2]** by calling it with either **<** or **>** as the left argument. **[3]** calculated **Ktau** and counts samples size(**n1**). **[4]** calculates significance level(**z**).

```
     ∇ Ktau←{⍝ c=concordant d=discordant
[1]        cd←{1=⍴⍵:0 ⋄ (+/(1↑⍵)⍺⍺ 1↓⍵)+∇ 1↓⍵}
[2]        c d←(<cd ⍵)(>cd ⍵)
[3]        tau←(c-d)÷(c+d) ⋄ n1←×/¯2↑⍳⍴⍵
[4]        tau,z←(3×tau×n1*0.5)÷(2×5+2×⍴⍵)*0.5}

      a1 Ktau a2          ⍝  so Ktau=(25-3)÷(25+3)=.7857
0.7857142857 2.721794126  ⍝  so Ktau=0.7857 z=2.7218
```

```
      ⍝ here is one more example
      b1← 1  2 3 4 5 6 7 8  9 10 11 12
      b2← 2  1 4 3 6 5 8 7 10  9 12 11
     ⍝ c=10 10 8 8 6 6 4 4  2  2  0   so c=60
     ⍝ d= 1  0 1 0 1 0 1 0  1  0  1   so d= 6
      b1 Ktau b2                      ⍝ thus Ktau=(60-6)÷(60+6)=.8182
0.8181818182 3.702917599  ⍝  so Ktau=0.8182 z=3.7029
```

Linear Regression: compute Best Fit line from raw data ****

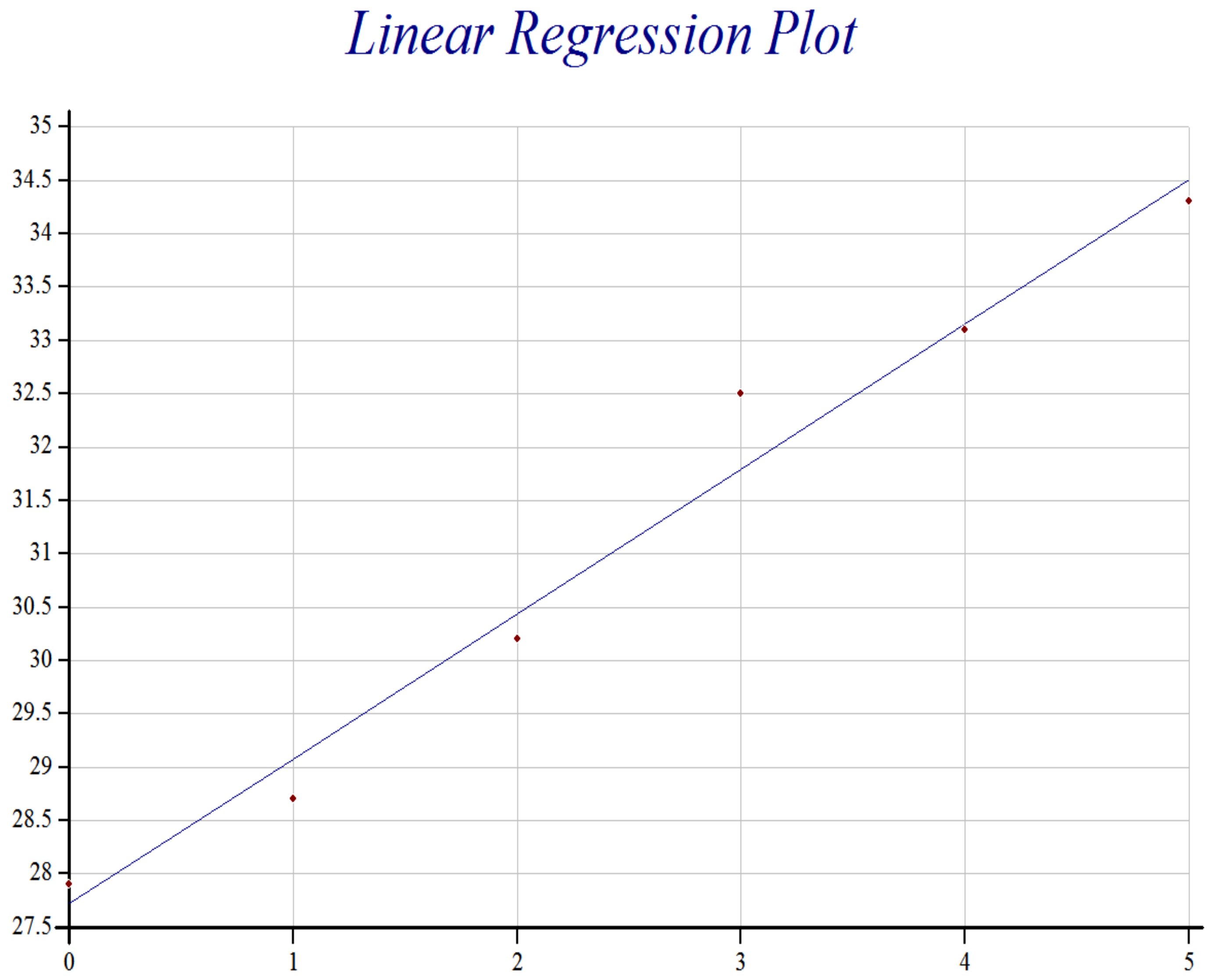
sd corr LinReg LinRegPlot
(see page 332 of Algebra 1 book) see also ch.Set 'Order'
Programs:**sd**:standard deviation **corr**:correlation **Reglin**:linear regression

```
 sd←{((+/(⍵-Mean ⍵)*2)÷¯1+⍴⍵)*0.5} ⍝ define program for standard deviation
 corr←{ma mw←Mean¨⍺ ⍵ ⋄ (+/(⍺-ma)×(⍵-mw))÷((+/(⍺-ma)*2)*0.5)×((+/(⍵-
mw)*2)*0.5)}                          ⍝ define program for correlation
 RegLin←{'y=ax+b  a=' 'b=' 'r=' 'r*2=',¨(⍵⌹⍺∘.*1 0),(⍺ corr ⍵)*1 2}
```

```
   R←x RegLinPlot y;yline;foot;a;b  ⍝ define linear regression plot
    ch.Set'Head' 'Linear Regression Plot'
    a b←y⌹⌽↑1,¨x                 ⍝ determine regression line formula
    yline←(a×x)+b                ⍝ get regression line points
    ch.Set'Footer'(x RegLin y) ⍝ get eq,r r*2 for footer label
    ch.Set'XYPLOT,GRID'          ⍝ set up the plot
    ch.Plot⍉↑x yline             ⍝ plot regression line
    #.ch.SetMarkers'Bullet'      ⍝ ch.∆markers shows other symbols
    ch.Scatter⍉↑x y              ⍝ data points as Bullets
    PG←ch.Close
    R←'View PG ⍝ to see it'
```

To create this fns type)ed **RegLinPlot** press enter & type above lines into editor.

```
        X←0 1 2 3 4 5                       ⍝ X raw data
        Y←27.9 28.7 30.2 32.5 33.1 34.3  ⍝ Y raw data
        X RegLinPlot Y                      ⍝ do regression
View PG ⍝ to see it
```



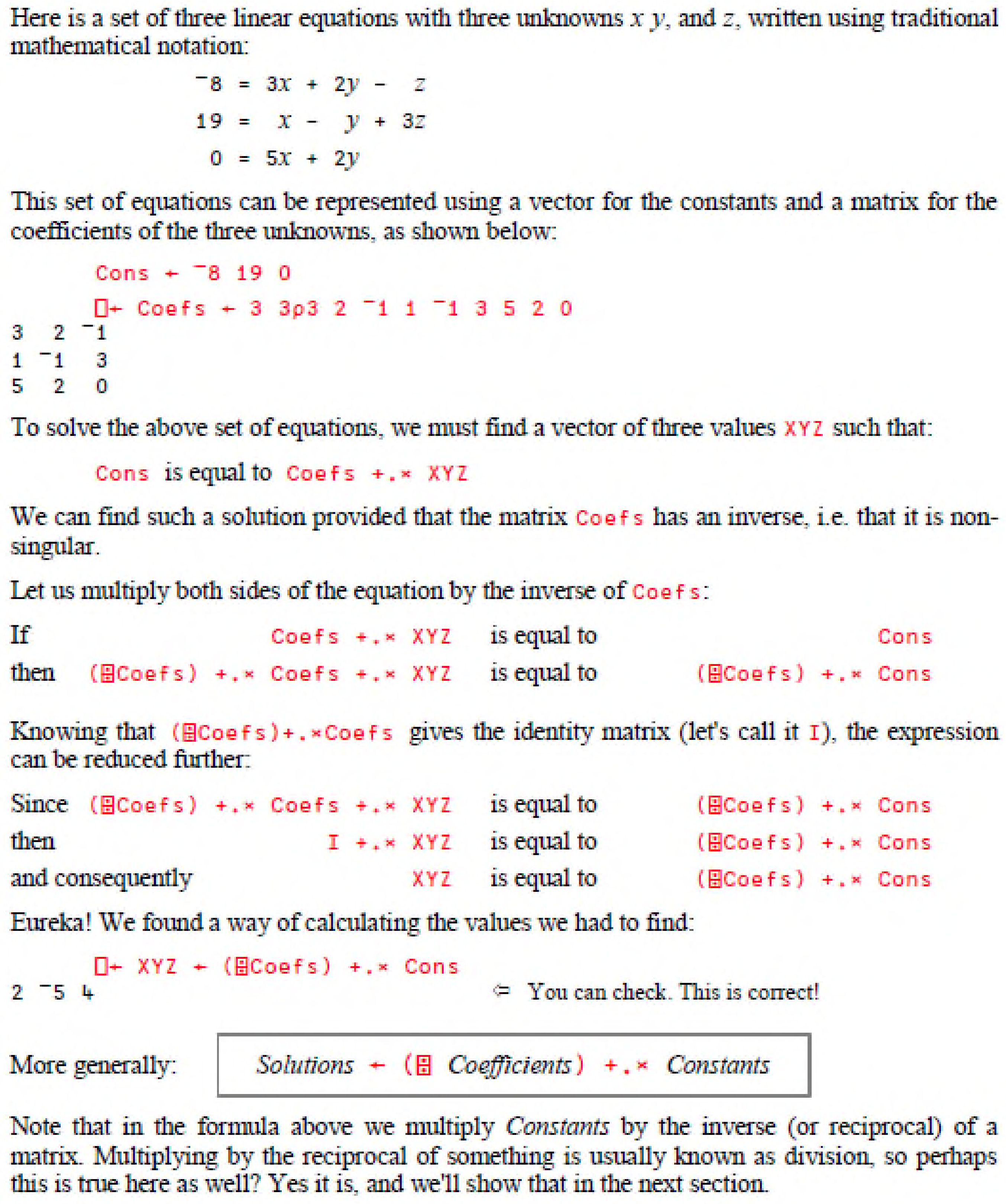
```
y=ax+b  a= 1.357142857  b= 27.72380952  r= 0.9881725632  r*2= 0.9764850146
```

So the above equation is: **Y=1.36X + 27.7** and correlation=.99

If you had only two points to plot this program would just find the perfect line equation between the two points and the correlation would be 1.0. The Domino (⌹) used in line [2] above is very powerful. It can be used to solve multiple regression problems where you are fitting multiple sets of data and nonlinear regression. It can also be used to solve sets of simultaneous equations.

Solve Set of Equations Easily with APL (Cons⌹Coefs) ****

Let me explain how **Cons⌹Coefs** in **APL** to solves simultaneous equations.



Here is a set of three linear equations with three unknowns *x y*, and *z*, written using traditional mathematical notation:

```
¯8 = 3x + 2y -  z
19 =  x -  y + 3z
 0 = 5x + 2y
```

This set of equations can be represented using a vector for the constants and a matrix for the coefficients of the three unknowns, as shown below:

```
      Cons ← ¯8 19 0
      ⎕← Coefs ← 3 3⍴3 2 ¯1 1 ¯1 3 5 2 0
3  2 ¯1
1 ¯1  3
5  2  0
```

To solve the above set of equations, we must find a vector of three values XYZ such that:

Cons is equal to Coefs +.× XYZ

We can find such a solution provided that the matrix Coefs has an inverse, i.e. that it is non-singular.

Let us multiply both sides of the equation by the inverse of Coefs:

If   Coefs +.× XYZ is equal to   Cons

then (⌹Coefs) +.× Coefs +.× XYZ is equal to (⌹Coefs) +.× Cons

Knowing that (⌹Coefs)+.×Coefs gives the identity matrix (let's call it I), the expression can be reduced further:

Since (⌹Coefs) +.× Coefs +.× XYZ is equal to (⌹Coefs) +.× Cons

then   I +.× XYZ is equal to (⌹Coefs) +.× Cons

and consequently   XYZ is equal to (⌹Coefs) +.× Cons

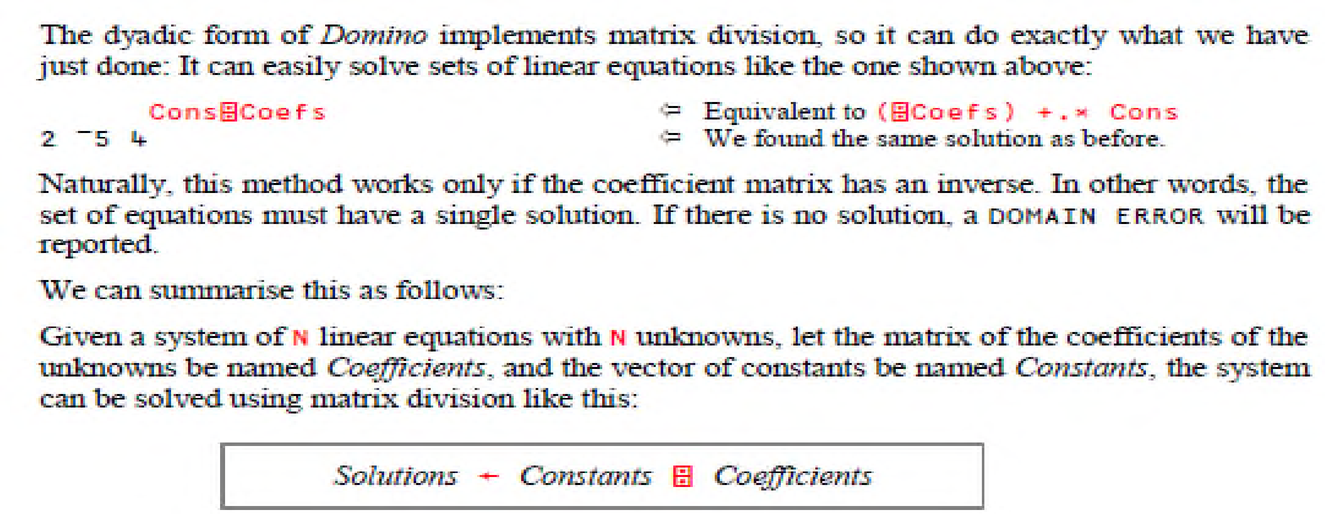
Eureka! We found a way of calculating the values we had to find:

```
      ⎕← XYZ ← (⌹Coefs) +.× Cons
2 ¯5 4                                ⇦ You can check. This is correct!
```

More generally:

$$\boxed{Solutions \leftarrow (⌹\ Coefficients)\ +.\times\ Constants}$$

Note that in the formula above we multiply *Constants* by the inverse (or reciprocal) of a matrix. Multiplying by the reciprocal of something is usually known as division, so perhaps this is true here as well? Yes it is, and we'll show that in the next section.



The dyadic form of *Domino* implements matrix division, so it can do exactly what we have just done: It can easily solve sets of linear equations like the one shown above:

```
      Cons⌹Coefs              ⇐ Equivalent to (⌹Coefs) +.× Cons
2 ¯5 4                        ⇐ We found the same solution as before.
```

Naturally, this method works only if the coefficient matrix has an inverse. In other words, the set of equations must have a single solution. If there is no solution, a DOMAIN ERROR will be reported.

We can summarise this as follows:

Given a system of N linear equations with N unknowns, let the matrix of the coefficients of the unknowns be named *Coefficients*, and the vector of constants be named *Constants*, the system can be solved using matrix division like this:

Solutions ← *Constants* ⌹ *Coefficients*

This exact same method of solving equations is also used for Regression Analysis upon which much of statistics is based. The Constants are the dependent variable and the Coefficients are the independent variables used to predict dependent variable. The Solutions is the prediction equation. The ⌹ operator does it all in APL from simple regression to multiple and nonlinear regression. Lets try a simple example first.

The Horse & Mule Problem[1] (WORDS TO ALGEBRA TO APL) ***

Here's a problem to translate from words to algebra to APL. A horse & a mule, both heavily loaded, were going side by side. The horse complained of its heavy load. "What are you complaining about?" replied the mule. "If I take 1 sack off your back, my load will become twice as heavy as yours. But if you remove 1 sack from my back, our loads will be the same." Now wise mathematician, 1st show me algebra then solve with APL for # sacks for horse and mule?" Use: **H=horse sacks** and **M=mule sacks**.

If I take one sack, (from horse=H)	H-1
my load (mule=M)	M+1
will be twice as heavy as yours.	1) M+1=2(H-1)
But if you take one sack from my back(M)	M-1
Your(H) load	H+1
will be the same as mine.	2) M-1=H+1

We have reduced the problem to a system of 2 equations in 2 unknowns:

1) M+1=2(H-1)	Now rearrange 1) & 2) for APL:	1) 3=2H+¯M
2) M-1=H+1	constants left & coefficients right	2) 2=¯H+M

Here's APL code from previous section: **Solutions←Constants ⌹ Coefficients**

```
3 2⌹2 2⍴2 ¯1 ¯1 1    ⍝ APL Constants=3 2 Matrix Coefficients=2 2⍴2 ¯1 ¯1 1
5 7                  ⍝ Solution: H(horse)=5 sacks and M(mule)=7 sacks.
```

So if mule took 1 sack from horse mule would have 8 & horse 4, and if mule gave 1 sack to horse they would both have 6 sacks.

[1] From **Algebra Can Be Fun** by Ya I Pearlman 1936

Linear Quad & Cubic Regression ***** reg←{x y←⍵ ⋄ y⌹x∘.*⌽0,⍺}

First some data for X and Y (used throughout the following examples)

```
      X←¯2 ¯1 0 1 2  ⋄ Y←0.25 0.5 1 2 4
```

```
      RegLin←{(⊂'y←(a×x)+b'),('a←' 'b←' 'r=' 'r*2=',¨⍕¨(⍵⌹⍺∘.*1 0),
        (⍺ corr ⍵)*1 2)} ⍝ define Linear Regression function(all one line)
      X RegLin Y          ⍝ call Linear Regression function
y←(a×x)+b  a← 0.9  b← 1.55  r= 0.93  r*2= 0.87 ⍝ linear results
```

```
      RegQuad←{(⊂'y←(a×x*2)+(b×x)+c'),('a←' 'b←' 'c←',¨⍕¨(⍵⌹⍺∘.*2 1 0))}
       X RegQuad Y                    ⍝ call quadratic regression function
y←(a×x*2)+(b×x)+c  a← 0.29  b← 0.9  c← 0.98   ⍝ quadratic results
```

```
      RegCube←{(⊂'y←(a×x*3)+(b×x*2)+(c×x)+d'),
        ('a←' 'b←' 'c←' 'd←',¨⍕¨(⍵⌹⍺∘.*3 2 1 0))} ⍝ (all 1 line again)
       X RegCube Y                    ⍝ call cubic regression function
y←(a×x*3)+(b×x*2)+(c×x)+d  a← 0.06  b← 0.29  c← 0.69  d← 0.98
```

Notice the similarity in the above 3 functions.

Regression coefficients	Equation	APL Domino Operator
Linear: a b	y←(a×x)+b	⍵⌹⍺∘.*1 0
Quadradic only	y←(a×x*2)+b	⍵⌹⍺∘.*2 0
Lin/Quadradic: a b c	y←(a×x*2)+(b×x)+c	⍵⌹⍺∘.*2 1 0
Lin/Quad/Cubic a b c d	y←(a×x*3)+(b×x*2)+(c×x)+d	⍵⌹⍺∘.*3 2 1 0

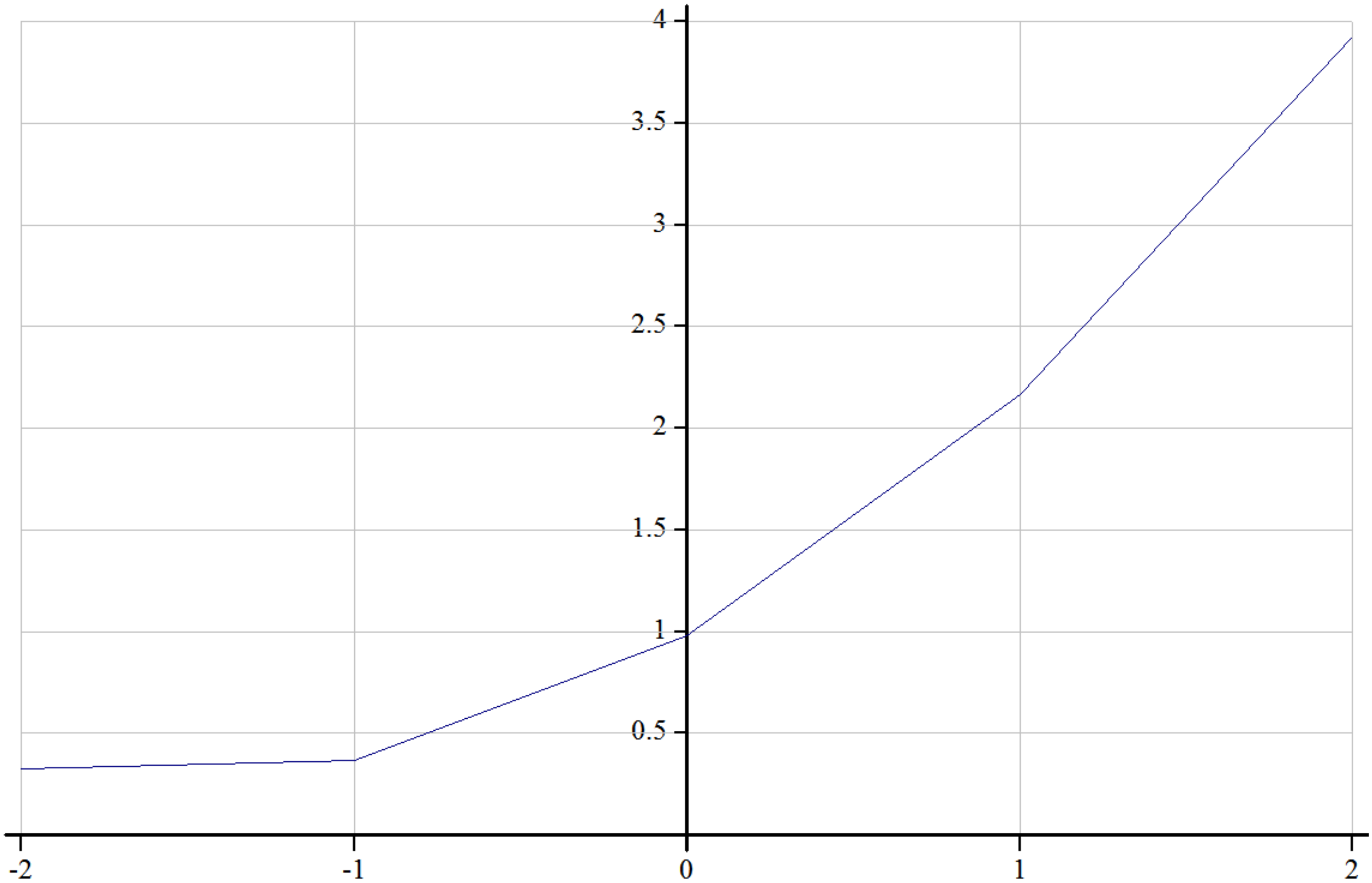
But there is a simpler way in APL. Since the 3 programs are so similar it is possible to write one function that can do linear quadric and cubic and actually it can go beyond cubic if you wish. Here is the function:

```
        reg←{x y←⍵ ⋄ y⌹x∘.*⌽0,⍺}    ⍝ does all types of simple regressions
       2⍕1 reg X Y                    ⍝ 1 is x^1 linear regression
0.9 1.55                              ⍝ a← 0.9  b← 1.55
       2⍕2 reg X Y                    ⍝ 2 is x^2 quadratic regression
0.29 0.98                             ⍝ a← 0.29  b← 0.98
       2⍕3 reg X Y                    ⍝ 3 is x^3 cubic regress
0.24 1.55                             ⍝ a← 0.24  b← 1.55
       2⍕¨(1)(1 2)(1 2 3) reg¨ ⊂X Y  ⍝ lin, lin & quad, lin & quad & cubic
0.9 1.55  0.29 0.9 0.98  0.06 0.29 0.69 0.98 ⍝ see 3 equations below
  ⍝ y=.9x+1.55    y=.29x^2+.9x+.98    y=.06x^3+.29x^2+.69x+.98
```

```
If you wanted little better labeling of these equations pass them to this:
RegEq function
      RegEq←{⍺←⍳¯1+⍴⍵ ⋄ 1↓⊃,/(⊂'+('),¨((⍕¨⍵),¨(⊂'×X*')),¨(⍕¨⌽0,⍺),¨')'}
      ↑ lab RegEq¨(lab←(1)(1 2)(1 2 3))reg¨⊂X Y No 2⍕ so show all decimals
(0.9×x*1)+(1.55×x*0)                                        ⍝ linear
(0.2857142857×X*2)+(0.9×X*1)+(0.9785714286×X*0)             ⍝ lin/quad
(0.0625×X*3)+(0.2857142857×X*2)+(0.6875×X*1)+(0.9785714286×X*0)⍝ lin/q/cub
```

Any 1 these equations could be easily cut, pasted & compared in **plotxy**.

```
plotxy X (Y←(0.2857142857×X*2)+(0.9×X*1)+(0.978571486×X*0))⍝ lin/quad plot
View PG ⍝ to see it
```



Actually any # of these equations could easily be plotted on the same plot. An example of plotting more than one equation on the same plot follows with automatic labeling of the lines also. All you have to do is enter your **x** range and your equations on line below beginning with **xandys.**

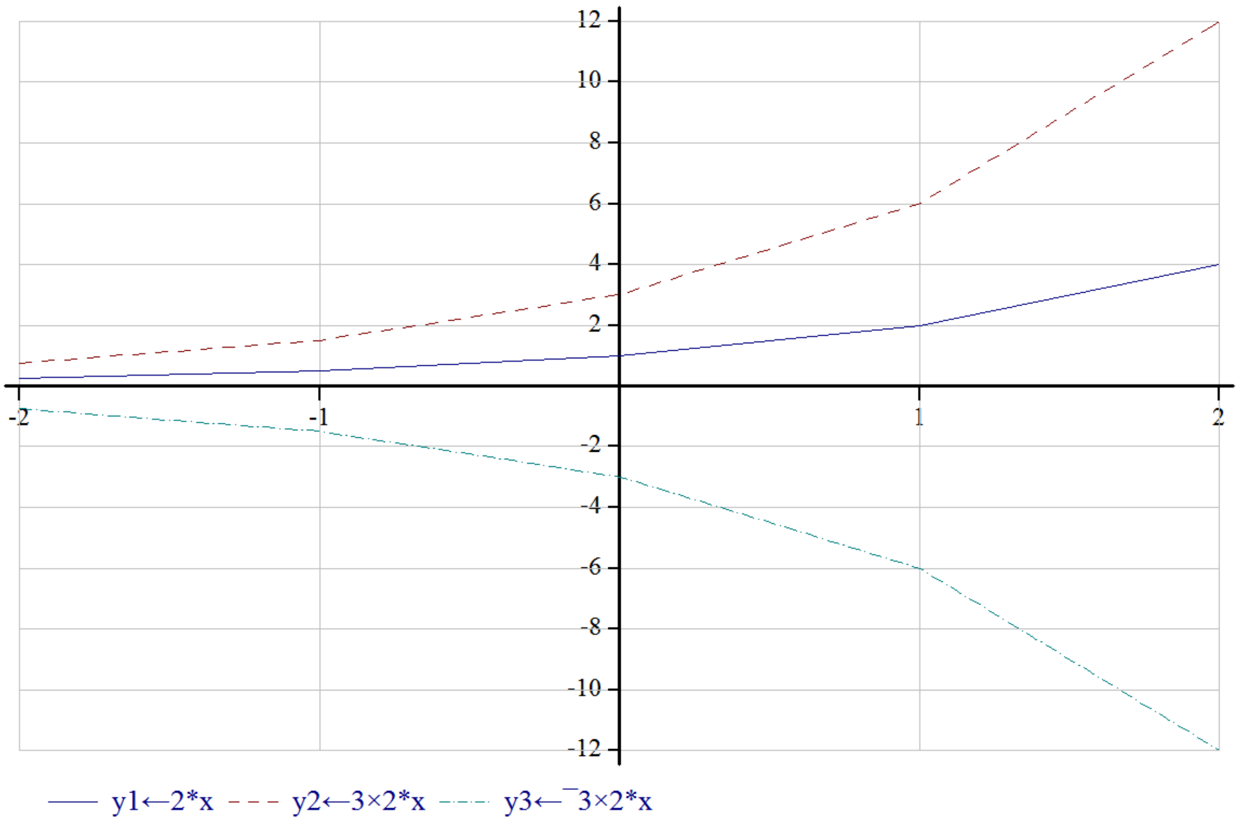
Plotting 3 Exponential Functions to Compare ****

Here is some data for three exponential functions from page 521 of **Algebra I** by McDougal Littell which I show you how to easily plot all at once.

X←¯3+⍳5	y1←2*X	y2←3×2*X	y3←¯3×2*X
¯2	0.25	0.75	
¯1	0.5	1.5	¯1.5
0	1	3	¯3
1	2	6	¯6
2	4	12	¯12

Here is how this data for X range of ¯2 to 2(X←¯3+⍳5) and equations: **y1 y2 y3** are computed inserting the X values into each of the equations (⍎¨**xandys**). The text equations are put into the key for display in the plot which is called in the 3rd line below(**plotxy**). This example plots 3 lines but any number of equations of any complexity to could be plotted.

```
      xandys←'X←¯3+⍳5'  'y1←2*X' 'y2←3×2*X'  'y3←¯3×2*X'
      ch.Set 'Key' (1↓⊃,/',',¨1↓xandys)
      plotxy ⍎¨xandys
View PG A to see it
```



Plotting in General in APL *

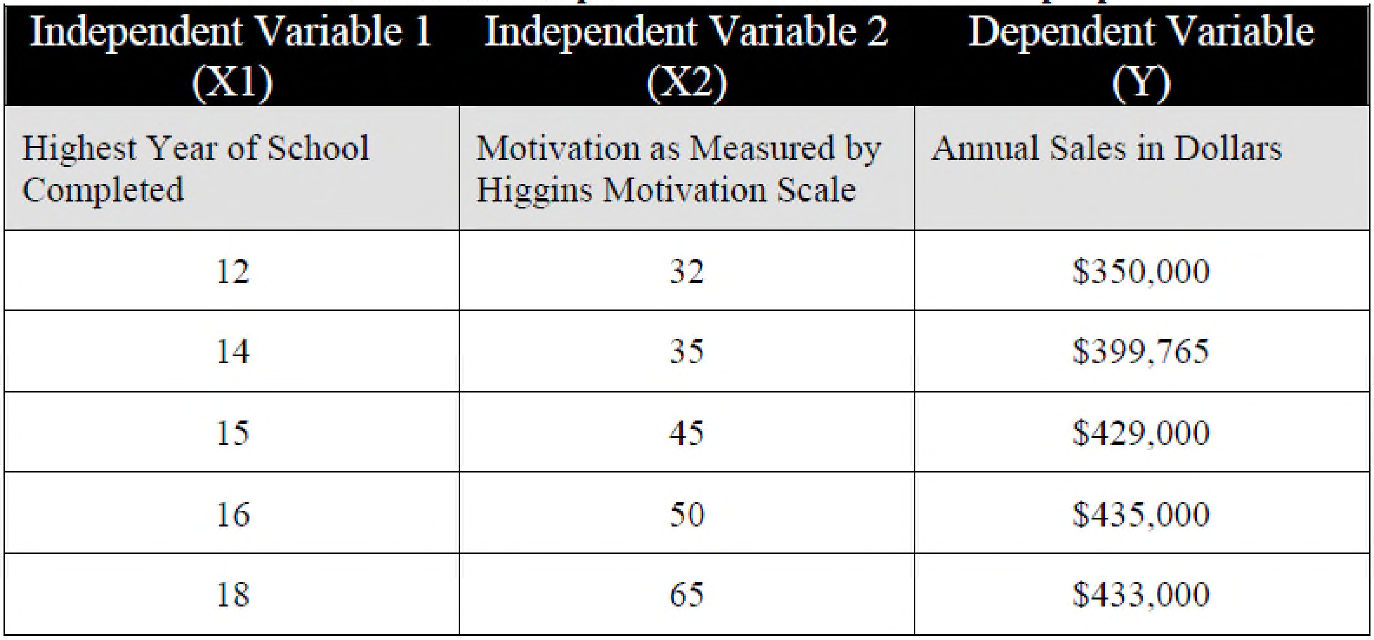
There is a very extensive plotting library which can do virtually any plot you want. In addition virtually everything can be customized. Fonts and colors can be changed, multiple axes are available, plots can be placed on top of each other, specific areas can be notated or colored etc. To see examples of all of the above and more simply click on each of the following commands. First load rainpro the press enter any of the lines below it.

```
    )load rainpro  ⍝ to load in the following graphics

    Samples.Slideshow 3 ⍝ Run through selected samples (with 3s delay)
    ActiveCharts.Active ⍝ Simple illustration of drawing a chart on a form
    ActiveCharts.Drill  ⍝ Sample drill-down application with Dyalog Gui
    ActiveCharts.Edit   ⍝ Sample data editor using draggable markers
```

Multiple Regression

In the previous examples there was one X variable, which predicted one Y variable. In the simultaneous equation examples there was one perfect solution. In the linear, quadratic and cubic models there was one X variable and we determined a best fit equation to predict Y. In multiple regression there will be a number of different X variables that are used together to find a prediction equation for Y. In multiple regression X is a matrix with different columns for different X variables all used to predict the Y variable. What different people will wear tomorrow depends upon many things such as their income, what they wore today, chance of rain, temperature, who they are trying to impress, how steep the mountain is etc. The calculation in APL is basically the same. Lets look at an example.

TABLE 4-1
Data Collected From Random Sample of 5 General Motors Salespeople



Independent Variable 1 (X1)	Independent Variable 2 (X2)	Dependent Variable (Y)
Highest Year of School Completed	Motivation as Measured by Higgins Motivation Scale	Annual Sales in Dollars
12	32	$350,000
14	35	$399,765
15	45	$429,000
16	50	$435,000
18	65	$433,000

Alien Attack *

The human flesh eating Martians are coming, but fortunately we have a very expensive ray gun, which can destroy their one giant saucer. Unfortunately the saucer is very elusive and the gun only destroys the saucer 1/3 of the time. Fortunately a high paid consultant suggested that the solution is to build 3 ray guns because 1/3 plus 1/3 plus 1/3 comes out to .9999 so 99.99% of the time the saucer would be destroyed. Unfortunately this is not correct. So we need your help to save the human race. If not 3 how many ray guns would be needed to be 95% certain to save the human race?. What about 99% certain? I was a little bit nervous about this and being wrong might have some huge negative consequences for us humans so I resorted to the Monte Carlo technique. The trick is to translate this into APL code.

If 3 guns fired randomly using ?3 3 3 APL returns 3 random numbers between 1 & 3. Using 1 for a hit & 2 & 3 for misses gives us our 1/3 for each gun.

```
       ?3 3 3
2 1 2  ⍝ so the second gun destroyed the saucer but I noticed no 3’s
```

So this looks dangerous to me. So we need some more checking. I only want to find 1’s so I modified my code a little.

```
      1=⎕←?3 3 3 ⍝ ⎕← assigns random #’s to output and then 1= matches
3 1 2            ⍝ these are the random gun shots show by ⎕←
0 1 0            ⍝ this shows that gun 2 was=1 and it destroyed the aliens
```

Now all I really care about is if 1 or more guns=1 and aliens are dead so I use ∨/ which, like +/ puts a plus between each number, ∨/ puts an or(∨) between each number and result is 1 if gun 1 or gun 2 or gun 3 = 1 otherwise ∨/ result is zero. So in examples below none of 3 3 2 = 1 so result is 0. But in 2nd example one or more of 1 2 1 = 1 so result is 1.

```
      ∨/1=⎕←?3 3 3
3 3 2                   ⍝ none of shots match 1
0                       ⍝ so saucer gets through and earth is lost
      ∨/1=⎕←?3 3 3
1 2 1                   ⍝ two shots=1
1                       ⍝ so saucer definitely destroyed
```

Now lets create a program and run it a few times and average the results.

```
      avg{∨/1=?⍵⍴3}¨1000000⍴3   ⍝ Remember avg←{(+/⍵)÷⍴⍵}.
0.704073                ⍝ so on average 3 guns kill saucer 70% of time.
```

The ¨1000000⍴3 makes up a million 3's which are passed one at a time using(¨) to the program to run 1 million times. ⍵ is the right argument to the unnamed function, The 3 is for 3 guns in this case so ⍵⍴3 becomes 3⍴3 which becomes 3 3 3. (avg{1∊?⍵⍴3}¨?1000000⍴3 uses membership(∊) works too)

Lets try 4 guns and our fns using membership(∊). Is 1 a member of ?4⍴3

```
      avg{1∊?⍵⍴3}¨1000000⍴4     ⍝ 4 guns each hit saucer 1/3 of time.
0.802378                        ⍝ 4 guns better but I want 95% or better.
```

Please figure out # guns needed to be 95% certain & let me know. Thanks!

Alien Attack Two *****

Wonderful we destroyed the saucer, but unfortunately the Martians came up with a new strategy. They built a zillion(more or less) small saucers. Fortunately they put an id number each saucer from 1 to N and we can see some of saucers coming and can read the id numbers on some of those. Unfortunately the id numbers are not in any particular order, they are random and further they are in binary not the base 10 we are used to. Fortunately APL has a built in function to change numbers from any base to and from base 10 and I have an idea of a way to estimate N from a sample (n) of random numbers from N.

First lets review number bases. In base 10 we have 10 digits for the first 10 numbers then we repeat using the same 10 digits like this:
0 1 2 3 4 5 6 7 8 9   10 11 12 13 14 15 16 17 18 19   20 21 22 23 etc
In binary we only have two digits 0 and 1 so the repeating happens faster.
0=0 1=1 2=10 3=11 4=100 5=101 6=110 7=111 8=1000 9=1001 10=1010 11=1011
In APL decode (⊥) and encode (⊤) do these conversions and more for us.

First let's decode(⊥) binary numbers to decimal.

```
        2⊥1 0             ⍝ binary  10 notice spacing of 1 0
2                         ⍝ decoded bin  10 decimal answer=2 (see above)
        2⊥1 0 0 1         ⍝ binary  1001
9                         ⍝ decoded 1001 decimal answer=9 (see above)
      2⊥¨(1 0)(1 0 0 1)   ⍝ do both numbers at once.
2 9
```

Decode(⊥) can work with other bases also for example days hours & minutes

```
      24 24 60⊥1 2 45    ⍝ convert 1 day 2 hour and 45 minutes to minutes
1605                     ⍝ (1day=24hr×60min)+(2hr×60)+45min = 1605 minutes
```

And here's an example assembling a decimal number from it's components:

```
10⊥1 3 5 2               ⍝ we have 1 thousand 3 hundreds 5 tens 2 ones
1352                     ⍝ which becomes 1352
```

Now let's see how encode(⊤) works:(Note ⊤ needs multiple left side #'s)

```
      10 10 10 10⊤1352   ⍝ break number 1352 back down into decimal parts
1 3 5 2                  ⍝ means:1 thousand 3 hundreds 5 tens 2 ones
     24 24 60⊤1605       ⍝ break 1605 minutes into days hours and mins
1 2 45                   ⍝ 1 day 2 hours and 45 minutes
```

Now one further thing before we get on to the problem. In converting a decimal number to binary we need to know how many 2's to put to the left of encode. The answer is: 1+the floor of the base 2 log of the number. In APL this is found like this.

```
   1+⌊2⍟9                ⍝ 1 plus floor(⌊)of base 2 log(⍟) of #
4                        ⍝ so 9 is 4 digit binary number(as we saw above)
   1+⌊2⍟1000000          ⍝ 1 million requires 20 digits.
20
```

So Here is how to do it for these two examples 9 and 1 million:

```
   ((1+⌊2⍟n)⍴2)⊤n←9
1 0 0 1                                 ⍝ 9 in binary 4 digits needed
   ((1+⌊2⍟n)⍴2)⊤n←1000000
1 1 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 ⍝ 1 million in 20 binary digits
```

We will be using this a bit so lets make it easy and write a function.

```
   Dec2Bin←{((1+⌊2⍟⍵)⍴2)⊤⍵}
```

Ok now lets get to work on the saucers. Lets assume there are 1 million saucers (N=1000000) and we get the id's for random 100 Saucers(n=100)

```
      N←1000000 ⋄ n←100    or N n←1000000 100 would also work
      id←Dec2Bin¨n?N  ⍝ get 100 random id's from million & make binary
```

Now here's the magic formula to predict N(the total # of saucers) from a sample. In this case we already know N so we can see how well it works. Here's the formula in conventional math notation: **Nest=(n+1)/n × Max(id)-1**

Now the equation in APL with the added conversion from binary to decimal.

```
      +Nest←(((n+1)÷n)×(⌈/2⊥¨id))-1  ⍝ Nest estimates N(total saucers)
996750.83                            ⍝ pretty close to a million
```

Note max is **⌈/** and **2⊥¨id** converts each bin **id** to decimal **id**. Finally an extra set of parentheses is needed as APL goes right to left with no order of operations rules so to make the subtraction(-1) goes last & rest needs parentheses around it.

Now lets put all this together in a function that we can play with to see if we can count the saucers with a smaller sample than 100. So lets put two functions together into a third function so what we are doing is clear.

```
    SaucEst←{(((1+⍴,⍵)÷⍴,⍵)×(⌈/2⊥¨⍵))-1} ⍝ est N ⍵=n random bin id's
    SaucDo←{SaucEst Dec2Bin¨⍺?⍵}         ⍝ generate and estimate N
    100 SaucDo 1000000          ⍝ call program ⍺=n(sample) ⍵=N(population)
984090.48                       ⍝ estimate from n=100(actual=1 million)
```

Now lets run it 500 samples of size 100 and get the average.

```
      avg 100{α SaucDo ω}¨500ρ1000000
999218.1065
      avg 100{α SaucDo ω}¨500ρ1000000
1000095.813
```

So on the average it is pretty much perfect. It's unbiased, there is no tendency to over or underestimate N. Now the other question we must answer is, "Is it efficient?". If we have to run it 500 times that is not too good. We will be stuck at our telescope for a long time. Lets run some smaller samples and see what happens but instead of average lets look at variability using the Standard Deviation:

```
      sdev←{((+/(ω-(ρω)ρavg ω)*2)÷(1↑ρω)-1)*0.5} ⍝ avg sum sq div by mean
      sdev 10{α SaucDo ω}¨500ρ1000000      ⍝ 500 samples size 10 each
87455.71697                                ⍝ standard deviation
      sdev 10{α SaucDo ω}¨500ρ1000000      ⍝ 500 samples size 10 each
91742.41377                                ⍝ standard deviation
      sdev 100{α SaucDo ω}¨500ρ1000000     ⍝ 500 samples size 100 each
9519.923639                                ⍝ standard deviation
      sdev 1000{α SaucDo ω}¨500ρ1000000    ⍝ 500 samples size 1000 each
1017.421925                                ⍝ standard deviation
      sdev 10000{α SaucDo ω}¨500ρ1000000   ⍝ 500 samples size 10000 each
100.0278067                                ⍝ standard deviation
```

So bigger samples are more accurate. Can you see an even more specific pattern? Lets divide SD by reverse of rounded sample sizes

```
     91742 9520 1017 100 11÷⌽10 100 1000 10000
0.91742 0.952 1.017 1                      ⍝ decreasing factor of ~10
```

That is interesting. Each time I increase sample size by a factor of 10 the variability decreases by a factor of ~10. Is this just chance? Lets test this theory by trying one more even larger sample. Since the last one took some time and I want try 100000 this time of course I will decrease the number of trials by a factor of 10 from 500 to 50.

```
      sdev 100000{α SaucDo ω}¨50ρ1000000 ⍝ 50 samples size 100000 each
10.60093385                              ⍝ very consistent Ssdev decrease
```

Lets check as we did before.

```
     91742 9520 1017 100 11÷⌽10 100 1000 10000 100000
0.91742 0.952 1.017 1 1.1 ⍝ Looks good
```

So to summarize the estimation equation is unbiased, It does not over or underestimate N. And if I increase the sample size by a factor of 10 the variability of my prediction decreases by a factor of ~10.

So if I saw saucers with the follow binary id numbers how many total Martian saucers is your best estimate.

1. (1 0 0 1)(1 1 0 0 0)(1 0 1 1)
2. (1 0 0 1)(1 0 1 1)
3. (1 0 0 0 1 1 0 0 0 1 1)(1 1 0 0 0)(1 0 1 1)(1 0 0 1 1 0 0 1 0 0)

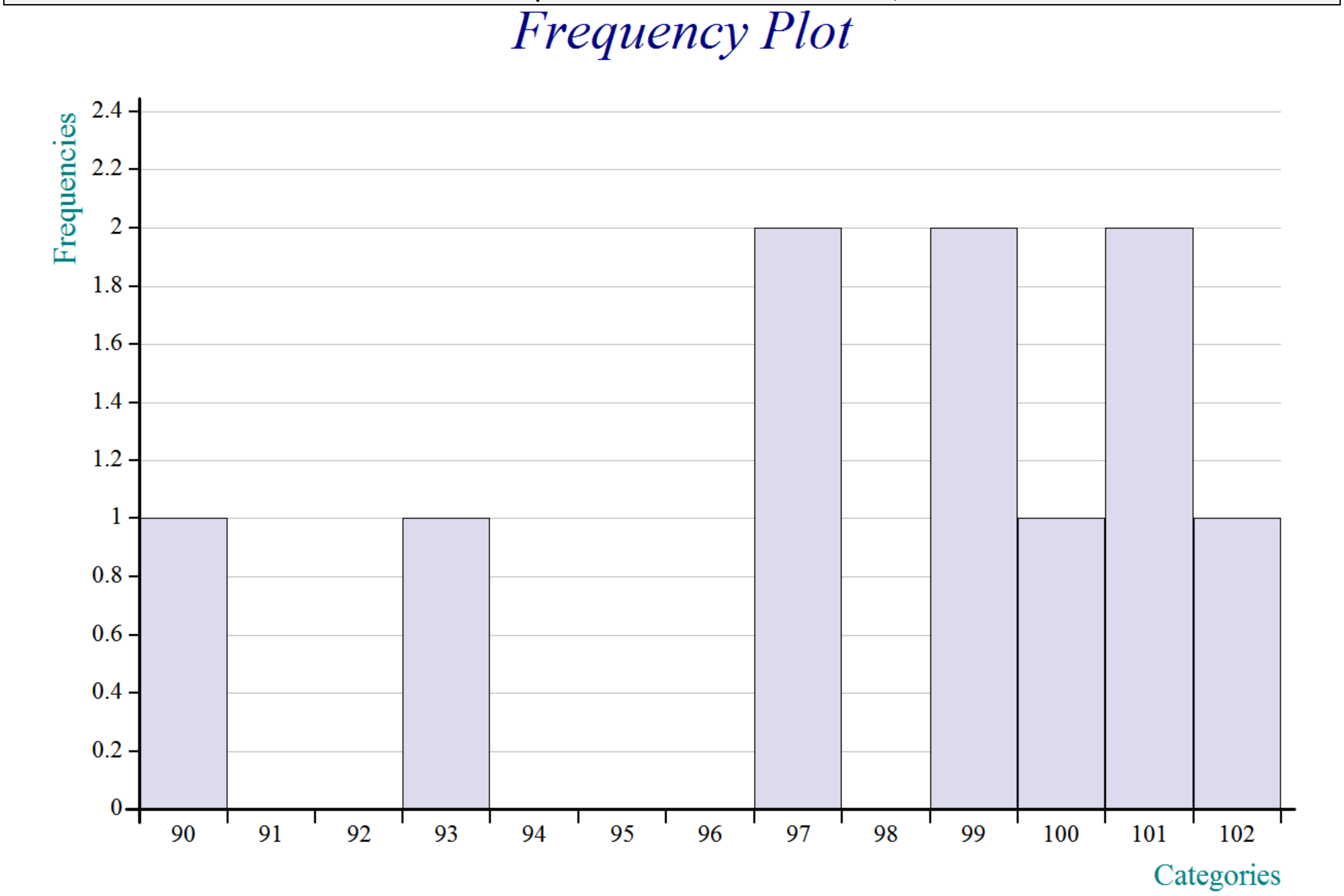
And extra credit: what are all the above binary id numbers in decimal form?

Answers:

```
1(9,24,11=31),2(9,11=15.5),3(1123,24,11,612=1402.75)
(SaucEst(1 0 0 1)(1 0 1 1))(2⊥¨(1 0 0 1)(1 1 0 0)(1 0 1 1))
```

Finally lets plot some data to see what variability looks like. So run it 10 times with each sample=20. The correct answer is 100 saucers. So 8 of 10 close but estimates of 90 & 93 are off. The avgerage is 97.9 Pretty good!

```
      ch.Set 'Footer' (Z←'FreqPlot ⊃¨20 SaucDo ¨10⍴100') ⋄ ⍎Z
```



FreqPlot ⊃¨20 SaucDo ¨10⍴100

How Often Will Current Year ÷ By Your Age Be Even? *

This example taken from:
http://www.mathgoespop.com/2010/01/a-mathematical-new-years-game.html

Lets say you are 16 and the current year is 2014. Lets find out if in any of the next 12 years your age divides evenly into the corresponding year.

```
      ages←15+⍳12
      years←2013+⍳12
```

Now we could just divide years÷ages & look, but let APL select for us. Compare **years÷ages** to floor(⌊)**years÷ages** to see where it's even. Floor(⌊) rounds down. So floor on an integer will = the number. For decimals this will not be the case. **2=⌊2** but **2.3≠⌊2.3** because **⌊2.3** is **2** and **2.3≠2**.

```
      (years÷ages)=(⌊years÷ages)
0 0 1 0 0 0 0 0 0 0 0 1 ⍝ 1=even result, else=0: (years÷ages)=(⌊years÷ages)
```

So 3rd and 12th years are even. Lets use these 1's and 0's to select those years:

```
      ((years÷ages)=(⌊years÷ages))/years
2016 2025
```

Or to see the ages:

```
      ((years÷ages)=(⌊years÷ages))/ages
18 27
```

Or with a little more fiddling both years and ages together:

```
      ↑(⊂(years÷ages)=(⌊years÷ages))/¨years ages ⍝ OR ↑(d=⌊d←÷/ya)/¨ya←years ages
2016 2025
18 27
```

Now lets create a program to do this and have it automatically check all ages from you current age to age 100. ⎕TS returns today's date and time and 1↑selects the first part of it which is this year. So α is set to the years from current to the year you will be 100. ω is input by you and should be your current age. The program has 2 lines separated by the ⋄. The first line sets up the ages and matching years and the second line does the selection

```
EvenYrDivAge← {α←(1↑⎕TS-1)+⍳⍴ages←ω+0,⍳100-ω ⋄ ↑(⊂div=⌊div)/¨α ages (div←α÷ages)}
```

Lets try the program now. Say you are 16 and the date today is 2015.

```
        EvenYrDivAge 15
2016 2020 2025 2040 2050 2080 2100
  16   20   25   40   50   80  100
 126  101   81   51   41   26   21
```

So there are 7 years that a person who is 15 in 2015 will have an age that divides the current year evenly. Those ages are 16, 20, 25, 40, 50, 80, and 100. The last row above shows the other factor of the division. So for example 16×126=2016.

Are all Numbers of Form abcabc Divisible by 13? ***

How can that be? Most numbers are not divisible by 13. Lets check it out.

```
      123123÷13    ⍝ 123123 follows the abcabc format: a=1 b=2 c=3
9741               ⍝ yes that one is
      264264 813813 547547÷13
20328 62601 42119  ⍝ yes those 3 are
```

Lets write a program to test this out more thoroughly with 3 little fns.

```
      rand3u←{ω?9} ⍝ make 3 unique random digits a b c with values 1-9
      dup2←{10⊥ω,ω} ⍝ duplicate a b c and smooshes them together: abcabc
      div13←{(⌊x)=x←ω÷13} ⍝ x is # ÷ 13. now see if round down (⌊) of x=x

      div13 dup2 rand3u 3     ⍝ test it. remember apl works right to left
1                             ⍝ 1 yes the abcabc # is evenly ÷ by 13
```

```
      div13 ⎕←dup2 ⎕←rand3u 3 ⍝ use output windows to see intermediates
2 5 8                         ⍝ 3 random digits made by rand3
258258                        ⍝ 3 digits duplicated and smooshed by dup2
1                             ⍝ # ÷ 13 & compare to #'s floor 1=div by 13
```

```
      div13¨ ⎕←dup2¨ ⎕←rand3u¨ 3 3  ⍝ try it twice using each (¨)
```

```
 9 5 9  5 3 7                       ⍝ the two different a b c’s
959959 537537                       ⍝ each duplicated & smooshed together
1 1                                 ⍝ each is evenly ÷ by 13
```

```
      +/div13¨ dup2¨ rand3u¨ 50000⍴3 ⍝ try 50,000 times & add up 1’s(+/)
50000                               ⍝ all 50,000 #’s were divisible by 13
```

What if a b & c are not unique #’s? For example is 111111 divisible by 13. Lets revise rand3u to allow non unique numbers & 0’s and try again.

```
      rand3←{¯1+?⍵⍴10} ⍝ creates random numbers that may not be unique
       rand3¨ 4⍴3
 2 1 8  6 1 1  0 6 1  5 8 5  ⍝ group 2 and 4 are not unique sets of #’s
                             ⍝ group 3 will be 5 digit # 61061
      +/div13¨ dup2¨ rand3¨ 50000⍴3 ⍝ try with possible non unique a b c’s
50000                               ⍝ 50,000 non unique are evenly ÷ by 13
```

What Is Your Name Worth? *

If each letter in alphabet was worth a different amount of points (A=1 B=2... Z=26, whose name would be worth the most points?

If A=1 B=2 C=3 . . . Z=26 then ABE would be worth 1+2+5=8 points.

In APL There is an system function ⎕A which returns the capital letters in the alphabet. In boxes below **boldface** is APL, rest after **⍝** is comments.

```
      ⎕A                      ⍝ type ⎕A into APL session
ABCDEFGHIJKLMNOPQRSTUVWXYZ    ⍝ and this comes back. Try it!
```

Now we can use dyadic index of(⍳) to find where in ⎕A different letters are in a name (must be capitals).

```
      ⎕A⍳'ABE'        ⍝ dyadic means 2: ⍳ has a left and right argument.
1 2 5                 ⍝ so positions in ⎕A for ABE are A=1 B=2 E=5
```

Now a fns to get index numbers of letters & then add them up using +/:

```
      NAMSUM←{+/⎕A⍳⍵} ⍝ note APL goes right→left: finds indexes then adds
      NAMSUM 'ABE'    ⍝ call fns NAMESUM pass it a name to index and add
8                     ⍝ So ABE’s score is 8
```

Lets try on few names: (again remember they must be all capitals)

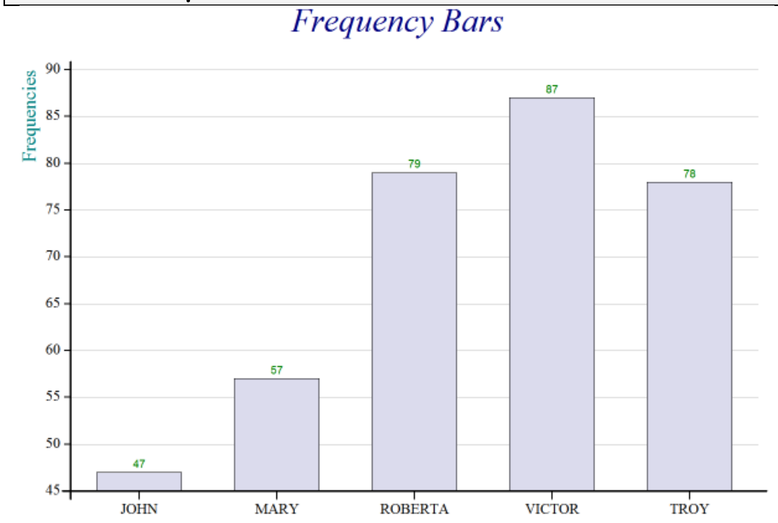
```
      NAMES←'JOHN' 'MARY' 'ROBERTA' 'VICTOR' 'TROY' ⍝ store names
      NAMSUM¨NAMES    ⍝ call fns NAMESUM for each(¨) of the names in NAMES
47 57 79 87 78        ⍝ so VICTOR the fourth name wins.
```

Lets make a labeled table & bar graph so we can see the results better:

```
    +DATA←↑(NAMES)( NAMSUM¨NAMES) ⍝ make table. The + shows the table
 JOHN  MARY  ROBERTA  VICTOR  TROY
   47    57       79      87    78
```

Now put cursor on **DATA** & click **Barchart icon** on toolbar at top or enter:

```
      FreqBar DATA
```



Rate Writing Based Upon Word And Sentence Length ***

First lets store some data in variable lincoln. Here is something he wrote:

> If we could first know where we are, and whither we are tending, we could then better judge what to do, and how to do it. We are now far into the fifth year, since a policy was initiated, with the avowed object, and confident promise, of putting an end to slavery agitation. Under the operation of that policy, that agitation has not only, not ceased, but has constantly augmented. In my opinion, it will not cease, until a crisis shall have been reached, and passed. "A house divided against itself cannot stand." I believe this government cannot endure, permanently half slave and half free.

There're many ways to read data into APL. In this case the easiest way is to use cut and paste, but text is too long so do this in two steps:.

```
       lincoln←'If we could'  ⍝ type this and press enter.
```

Now open up edit window by double click on word **lincoln** & cut & paste above text into window. Incidentally the edit window is very useful to add, change, delete or just look at any information in any variable or program you have already created. All you have to do is double click on its name.

First a fns to elim unneeded punctuation, but keep sentence end stuff .?!

```
      elim←{(~ω∊α)/ω}          ⍝ fns to eliminate α chars from ω
      sam←',;:"'elim lincoln  ⍝ eliminate (,;:") from lincoln and store in sam.
Notice that .?! are not in the list, so .?! will be left in for now.
```

Now fns to partition character strings into either words or sentences, default partition by spaces(α←' '), so each partition contain 1 word so we can count word length with (ρ). Keep program flexible so can also partition by sentence end markers(α='.?!') so we can also count words in sentences.

```
      partition←{α←' ' ⋄ ⎕ML←3 ⋄ (~ω∊α)⊂ω}⍝ partition fns (default is words)
      ρsentence←'.?!' partition sam   ⍝ sentence contains the sentences
6                                     ⍝ the ρ displays that there are 6 sentences
```

Now two fns: one to average word length & one to average number of words per sentence in Lincoln’s speech. Steps: 1)eliminates all punctuation including .?! first, then 2)partitions into words using the default spaces, then 3)finds the size of each word and then 4)averages those sizes.

```
      avgwordlen←{avg ρ¨partition ',;:.?!"'elim ω} ⍝ elim .?! also here
      avgwordlen lincoln
 4.447619048                             ⍝ average word length for whole doc
```

Next fns 1)eliminates punctuation except .?! then 2)partitions into sentences using .?! and then 3)partitions each sentence into words using spaces, then 4)finds the number of words within each of the sentences(⍴¨) then 4)exposes(⊃,/) the word counts for each of the sentences and then 5)finds the average number of words for the sentences.

```
      avgsentlen←{avg ⊃,/⍴¨ partition ¨'.?!' partition ',;:"'elim ⍵}
      avgsentlen lincoln
17.5                                    ⍝ average words per sentence
```

Now compare Lincoln and Shakespeare, using some text from Romeo and Juliet.

```
      avgwordlen¨ lincoln romeo
  4.447619048    4.135231317      ⍝ so Lincoln uses very slightly longer words
      avgsentlen ¨ lincoln romeo
17.5 21.61538462                  ⍝ but Shakespeare sentences are ~4 words longer
```

Stylometry: The analysis of text documents *****

Stylometry is often used to attribute authorship to anonymous or disputed documents. It has legal, academic & literary applications, ranging from ? of authorship of Shakespeare's works to forensic linguistics. (Wikipedia)

I will show some APL functions I created to analyze and compare different authors. In section below I analyze/ compare first 6 chapters from Mark Twain's Huckleberry Fin with 3 chapters from Mary Shelley's Frankenstein.

```
    )load Anna3     ⍝ to access the Stylometry Fns first load Anna3
    )cs Stylometry  ⍝ then change to Stylometry namespace(subfolder)
```

A Good text data source *www:gutenberg.org* .Project Gutenberg offers 45,263 free ebooks to download. The easiest way to download is to go to Gutenberg.org, find a .txt version of a book and display it. Then cut and paste sections or chapters into character variables in Anna3.Stylometry like this:

```
TwainHuckFin1←'xxx'  ⍝ enter a line with your variable name
                     ⍝ now double click on TwainHuckFin1
                     ⍝ delete xxx and paste text in and press ESC
)save                ⍝ now save it.
```

Then I put 9 chapter names in var called **Txts**. [see **Txts** in **VARS** section]

Once I have saved my sample text files, then I choose ways to compare them:

1. Compare average sentence length. (use APL: **AvgSentLen**)
2. Compare average word length. (use APL: **AvgWordLen**)
3. Compare vocabulary level. (use APL: **VocLevel** using 32 levels of Dunn-Rankin vocabulary test **L1-L32**)
4. Compare percentage of function words used. (use APL: **PercentWords** and file **FUNCTIONWORDS**(321 common function words).
5. Compare percentages of positive and negative words used. (use APL: function **PercentWords** with files **PosWords**(114) and **NegWords**(141).

1. Lets compare average sentence length for these 9 chapters: 6 from Twain and 3 from Shelly. Shelly’s sentences tend longer, but it is not clear cut.

```
            2⍕AvgSentLen¨⍎¨Txts
```

```
 18.43 15.55 19.35 14.05 14.53 19.98 21.67 23.65 19.43
```

2. Lets compare average word length for these 9 chapters: 6 from Twain and 3 from Shelly. It looks like Shelly’s words are consistently longer with Twain always in low 4’s and Shelly always in the low 5’s.

```
         2⍕AvgWordLen¨⍎¨Txts
 4.20 4.30 4.29 4.15 4.28 4.14 5.16 5.19 5.11
```

3. Let’s compare Vocabulary level for these 9 chapters: 6 from Twain and 3 from Shelly. It looks like Shelly’s vocabulary level is much lower than Twains except for Twain’s chapter 4 which was lower than all of Shelly’s.

```
          2⍕VocLevel¨⍎¨Txts
 11.70 14.35 10.55 3.43 11.25 11.75 5.44 5.08 6.26
```

4. Let’s compare percentage of **FUNCTIONWORDS** for these 9 chapters: 6 from Twain and 3 from Shelly. **FUNCTIONWORDS** is a variable of 321 words useful in detection of different people's styles. Function words are the words we use to make our sentences grammatically correct. Pronouns, determiners, and prepositions, and auxiliary verbs are examples of function words. Words such as: a, about, and, as, my, she, almost, before, and except are all function words. *http://myweb.tiscali.co.uk/wordscape/museum/funcword.html*
Shelly’s use of function words is consistently much lower than Twains.

```
        2⍕PercentWords¨⍎¨Txts
 59.82 59.09 59.50 57.12 58.56 60.02 52.38 51.81 53.13
```

Now one APL fns computes & labels all 6 Stylometrics(stored in variable **Fnames**) for the 9 chapters(stored in **Txts**). [see **VARS** section below]

```
        Fnames StyTbl Txts
Text\Fns     AvgSentLen AvgWordLen VocLevel FunctionWords PosWords NegWords
TwainHuckFin1    18.19        4.20      11.70    59.13         .76     1.90
TwainHuckFin2    15.55        4.30      14.80    58.31         .37      .73
TwainHuckFin3    19.35        4.29      10.55    58.86         .26      .64
TwainHuckFin4    14.05        4.15       3.43    56.49         .63     1.33
TwainHuckFin5    14.41        4.28      11.25    56.82         .54     1.01
TwainHuckFin6    19.98        4.14      11.75    59.15         .69     1.09
Frankenstein1    21.67        5.16       5.44    52.38         .74      .50
Frankenstein2    23.65        5.19       5.08    51.81        1.04      .25
Frankenstein3    19.43        5.11       6.26    53.13         .45      .53
```

---**VARS** --(USED BY FNS. In Anna3.Stylometry)------------------------

```
SYMB← '.?!`~@#$%^&*()_+-=[{]}\|;:",<>/0123456789'
```

FUNCTIONWORDS(321) words provide sentence structure but limited meaning. Some examples follow:

```
       a  about  above  after  again  ago  all  almost  along  already
       although always  am  among an  and  another  any  anybody
```

PosWords(114) words like:free easy lucky. **NegWords**(114) like:bad sad hurt

```
Txts←((⊂'TwainHuckFin'),¨⍕¨⍳6),((⊂'Frankenstein'),¨⍕¨⍳3)      ⍝ clever way

Fnames←'AvgSentLen' 'AvgWordLen' 'VocLevel' 'PercentWords'    ⍝ easy way

Fnames,←'PosWords PercentWords' 'NegWords PercentWords'
```

---FNS (in Anna3.Stylometry) ---

```
AvgSentLen←{avg⊃,/⍴¨partition¨(3↑SYMB)partition(3↓SYMB)elim ⍵}

AvgWordLen←{avg⊃,/⍴¨partition SYMB elim ⍵}

VocLevel←{avg(⊃,/⍴¨(⍎¨'L',¨⍕¨⍳32)FindWords¨⊂⍵)/⍳32}

PercentWords←{⍺←FUNCTIONWORDS ⋄ ⊃100×(⍴⍺ FindWords ⍵)÷⍴partition ⍵}

StyTbl←{⍉((1,1+⍴⍵)⍴(⊂'Text\Fns'),⍵)⍪(((⍴⍺),1)⍴⍺),((⊂8 2)⍕¨⍎¨⍺∘.,(' ',¨⍵))}
```

---**UTILITY FNS** (in Anna3.Stylometry) ------------------------------------

```
avg←{(+/⍵)÷⍴⍵}      ⍝ average =sum of #s(+/) divided by(÷) # of numbers(⍴)
FindWords←{⍺←FUNCTIONWORDS ⋄ ((words)∊⍺)/words←partition case SYMB elim ⍵}

elim←{(~⍵∊⍺)/⍵}                               ⍝ elim unneeded SYMBOLS ⍺

partition←{⍺←' ' ⋄ ⎕ML←3 ⋄ (~⍵∊⍺)⊂⍵} ⍝ brk txt by ⍺ (ie spaces or periods)
  case←{res←⍵ ⋄ ⍺←0 ⍝ default low case ⍺:0=up2lower change ⍺=1=lower2up
      To From←{⎕UCS(⎕UCS ⍵)+¯1+⍳26}¨⍺⌽'aA'    ⍝ find 26 lower & uppers
      (bool/res)←To[From⍳(bool←⍵∊From)/⍵]    ⍝ change only letters up/down
      res                                     ⍝ return modified string res
    }
```

Four Fun With Numbers *****

The follow are 4 fun/amusing math/number problems & their solution in APL. Have fun and remember after you execute a line you can either put the cursor on any of the variables created to see what they look like or type their name on a line to see them displayed.

Find all 3 digit whole positive numbers whose digits are the same when added or multiplied together.

Remember Encode(⊤) can be used to break numbers into digits like this:

```
      10 10 10⊤126          ⍝ to break up one 3 digit decimal number
1 2 6
      (⊂10 10 10)⊤¨34 126  ⍝ & this to break up 2 or more numbers at once
 0 3 4  1 2 6
```

So here is the solution:

```
      ((+/¨d)=(×/¨d←(⊂10 10 10)⊤¨n))/n←99+⍳900
123 132 213 231 312 321
```

Find two positive numbers that have a 2 digit answer when their digits are added together and a 1 digit answer when digits are multiplied together.

```
      ((10≤+/¨d)∧(10>×/¨d←(⊂10 10)⊤¨n))/n←9+⍳90
19 91
```

Find all two 2 digit whole positive numbers that have same answer when their digits are multiplied together as when digits are divided by each other.

```
      ((÷/¨d)=(×/¨d←(⊂10 10)⊤¨n))/n←9+⍳90      ⍝ seems logical but fails
DOMAIN ERROR                                   ⍝ can't divide by zero so.
```

So here is the fix. Need to turn digits around so 20 30 become 02 and 03 etc. the reverse symbol(⌽) will do this.

```
      (b←(÷/¨d)=(×/¨d←⌽¨(⊂10 10)⊤¨n))/n←9+⍳90 ⍝ flip(⌽) each(¨) set digits
10 11 12 13 14 15 16 17 18 19 20 30 40 50 60 70 80 90
```

Find a 10 digit number containing each digit once, so that the number formed by the last n digits is divisible by n for each value of n from 1-10. For an easy example lets try 3 digits: 168 works because 1÷1, 16÷2, and 168÷3 all are evenly divisible with no residues(|) or decimal parts.

Let's break this problem into steps we have to do:

1) get some unique random digits 0-9, (each digit only once)

2) break digits up in increasing pieces (1 16 168)

3) do the divisions by 1,2,3,..10, (1÷1, 16÷2, and 168÷3)

4) check if the answers have no remainders(residues(|)), and

5) make a function repeatedly call it until it finds an answer.

Now fiddle on your own, then look below at my 5 steps to the solution.

1) I can imagine at least two different ways to get the random numbers.

```
      ⍕10⊥¯1+3?10  ⍝ 3UniqueRand#1-10, -1(0-9), squish, make # to char
879
```

OR even easier way using built in ⎕D which ='0123456789' as characters.

```
   ⎕D[3?10]        ⍝ 3 rand# with no replacement indices of ⎕D
251
```

2) Now break the char string into increasing pieces: '2' '25' '251'

```
      1 2 3↑¨⊂'251'  ⍝ take(↑) each(¨) of 1 2 3 on enclosed(⊂)'251'
2 25 251           ⍝ more general way (⍳3)↑¨⊂'251'
```

3) now do the divisions: actually find the residues or remainders (|).

```
      (⍳3)|⍎¨(⍳3)↑¨⊂'251'
0 1 2                         ⍝ so remainders for 2÷1=0 25÷2=1 251÷3=2
```

4) Now check to see if each remainder(|) =0

```
      0=(⍳3)|⍎¨(⍳3)↑¨⊂'251'    ⍝ if remainder=0 result of(0=) is true=1
1 0 0                         ⍝ so yes,no,no  for 2÷1=0 25÷2=1 251÷3=2
```

Now check to see if all remainders =0 (1=yes the remainder=0)

```
      ∧/0=(⍳3)|⍎¨(⍳3)↑¨⊂'251'  ⍝ so check if all(∧/) 1's(remainders=0)
0                             ⍝ so no all remainders are not 0
```

5) Create function to do all the above

```
      NDigit←{∧/0=n|⍎¨(n←⍳⍵)↑¨⊂c←⎕D[⍵?10]:⎕←c} ⍝ ⍵ is input ie 3
```

The fns is inside of **{}**. It's name is **Ndigit**. The code to the left of the **:** is called the guard. If the guard is true(1) The code to the right(**⎕←c**) with be executed. In this case the passing number(**c**) will be displayed. If the guard is false, the code to the result will not be executed and nothing will be displayed. Now lets try it 10 times for the 3 digit number.

```
      NDigit ¨10⍴3       ⍝ try 10 random 3 digit #'s. It finds 5 #'s
789                     ⍝ so residuals all=0: 7÷1 78÷2 789÷3
801                     ⍝ so residuals all=0: 8÷1 80÷2 801÷3
984                     ⍝ so residuals all=0: 9÷1 98÷2 984÷3
```

```
963                        ⍝ so residuals all=0: 9÷1 96÷2 963÷3
024                        ⍝ so residuals all=0: 0÷1 02÷2 024÷3
```

Now lets try the real problem with 10 digits. Warning there is only one correct number and there are many numbers to test so it will take a lot of runs. On my computer it took a number of minutes to find the 1 number. You might work your way up from 1000 10 digit numbers using **NDigit ¨1000⍴10**. Good luck. Tell me when you find it (hidden answer=4138006086-321458796).

Extra credit. If you think about it a bit, you may be able to eliminate some numbers and design a fns that runs faster by selecting only certain random numbers. Think for a minute and only then read my next sentence that will give you one such hint. OK here is my hint. The last digit is the tenth digit and that longest number must be divisible by ten and the only numbers that are divisible by ten end in zero so that is what the last digit must be. So in this case you could simply search for a nine digit number using the numbers one through nine and then tack a zero on the end. This should speed things up considerably. Can you create a special fns called **NDigit10** which will only do the 10 digit problem. The **NDigit** fns above is of course more general and will do all problems from 1 to 10.

There are constraints on other digits also which could be used. There’s a trade off as it will take more code and thought on your part but it will strain the computer less. Best to allocate resources between your brain & your computers brain to get job done most efficiently. You have a powerful partner but you have skills it does not have. Together the two of you can go very far. Alone neither of you will probably amount to a hill of beans.

How Many Draws To Get An Ace? ****

The following fns shows average # of draws to get an ace. The answer is unexpected. The fns shows its lines of code as it runs. Here is the fns:

```
FirstAce;S;first;avg
 'The First Ace Problem from Fifty Challenging Problems in Probability'
 ' by Frederick Mossteller 1965 Harvard University'
 S←{⎕←ω ⋄ ⍎ω} ⍝ utility to both show and execute a line
 'What is average number of cards to draw before getting an ace?'
 S'4?52 ⍝ The positions of four aces randomly placed in deck of 52 cards?'
 S'⌊/4?52 ⍝ Find 1st(min) of 4 new random ace positions in deck of cards.'
 S'first←{⌊/ω?52} ⍝ Turn above code to fns to find first position of ace.'
 S'first 4 ⍝ Call it once to find position of first ace.'
 S'first ¨10⍴4 ⍝ Call 10 times, find position of 1st ace in 10 shuffles.'
 S'avg←{(+/ω)÷⍴ω} ⍝ Write fns to average results.'
 S'avg first ¨500000⍴4 ⍝ Call it 500,000 times and average results.'
 'So ~10.6 cards to draw to get an ace on the average.'
 'More than 500,000 fills the workspace so here is a little workaround.'
 S'avg{avg first ¨500000⍴4}¨10⍴0 ⍝ Avg of 500,000 10 times and avg that.'
 'Notice these details in the code:'
 ' 1: Unnamed fns: {avg first ¨500000⍴4} as called only once inline.'
 ' 2: The 10 zeros: (10⍴0) not used. They only make fns run 10 times.'
 ' 3: 10.6 probably not your bet to be average # of draws to get an ace.'
 ' 4: Can you modify fns to see avg # of draws to get any spade?'
 ' 5: Can you simplify fns to get avg # of throws of dice to get a 3 is?'
```

And here is the fns both running and showing all its code:

```
      FirsAce
The First Ace Problem from Fifty Challenging Problems in Probability
 by Frederick Mossteller 1965 Harvard University
What is average number of cards to draw before getting an ace?
4?52 ⍝ The positions of four aces randomly placed in deck of 52 cards?
48 20 51 5
⌊/4?52 ⍝ Find 1st(min) of 4 new random ace positions in deck of cards.
17
first←{⌊/⍵?52} ⍝ Turn above code to fns to find first position of ace.
first 4 ⍝ Call it once to find position of first ace.
2
first ¨10⍴4 ⍝ Call 10 times, find position of 1st ace in 10 shuffles.
13 2 19 13 22 27 20 16 8 17
avg←{(+/⍵)÷⍴⍵} ⍝ Write fns to average results.
avg first ¨500000⍴4 ⍝ Call it 500,000 times and average results.
10.59594
So ~10.6 cards to draw to get an ace on the average.
```

More than 500,000 fills the workspace so here is a little workaround.

```
avg{avg first ¨500000⍴4}¨10⍴0 ⍝ Avg of 500,000 10 times and then avg those 10 averages.
 10.5960054
```

Notice these details in the code:
 1: Unnamed fns: {avg first ¨500000⍴4} If you wanted to use it more you should name it.
 2: The 10 zeros: (10⍴0) not used. They only make unnamed fns run 10 times.
 3: 10.6 is probably not your bet to be the average # of draws to get an ace.
 4: Can you modify fns to see avg # of draws to get any spade?
 5: What would you do to get avg # of throws of dice to get a 3?
 6: Can you determine avg # draws to get both a 4&5 ?

Five Card draw Probabilities ****

1. If draw 5 cards what is probability of 1,2,3 or 4 aces?

```
  {(+/{(+/(52↑4⍴1)[5?52])∊ ⍳4}¨⍵⍴0)÷⍵}1000000      ⍝ 1,2,3 or 4(∊⍳4)
0.34085  ⍝ ~ 34% for 1-4 aces if 1 million random deals
```

Now lets look at 1 or 2 or 3 or 4 aces individually:

```
    {(+/{(+/(52↑4⍴1)[5?52])∊ 1}¨⍵⍴0)÷⍵}1000000      ⍝ 1 ace(∊ 1)
0.29966                                             ⍝ 30% chance 1 ace
```

```
    {(+/{(+/(⊃,/4 48⍴¨1 0)[5?52])∊ 2}¨⍵⍴0)÷⍵}1000000 ⍝ 2 aces(∊ 2)
0.039946                                            ⍝ ~4% chance 2 aces
```

```
    {(+/{(+/((4⍴1),(48⍴0))[5?52])∊ 3}¨⍵⍴0)÷⍵}1000000 ⍝ 3 aces(∊ 3)
0.00171                                             ⍝ ~.1% chance 3 aces
```

```
    {(+/{(+/(4 48/1 0)[5?52])∊ 4}¨⍵⍴0)÷⍵}1000000    ⍝ 4 aces(∊ 4)
0.000016                                           ⍝ ~.002% chance 4 aces
```

Here is how it works. 4⍴1 makes 4 ones. 52↑ takes the 4 ones and pads with 48 zeros. (There are many ways to do this as I demonstrate above in each example.)

Ones will be the hits and zeros the misses. 5?52 takes 5 random numbers between 1 and 52 without replacement.

The random numbers are used to index the 52 1's and 0's generated. If the the random index numbers are between 1 and 4 a 1 will be selected(an ace) otherwise it is not one of the first four aces and a 0 will be picked.

These 5 selections(1 for each ace and 0 otherwise) are summed up and to see if they are a member of (ϵ). The 1 million results(0=no 1=yes) are then again summed and divided by the 1 million. Note there is a fns {} inside another fns{} The inner fns runs 1 million times summing up the times correct # aces are found in a million tries. The outer fns divides the sum by 1 million to get the percent. Another note ¨ωρ0 passes 0 to the inner fns 1 million times. The 0 is not used in the inner fns, it just causes the inner fns to run 1 million times spewing out a 1 or 0 each time that is then summed (+/) and divided by 1 million.

2. If draw 5 cards what is prob of 3,4,5 in a row of same suit.

```
    +cards←(52ρι13)+(13/0 20 40 60) ⍝ create deck 4 suits 13 cards in each
1 2 3 4 5 6 7 8 9 10 11 12 13 21 22 23 24 25 26 27 28 29 30 31 32
      33 41 42 43 44 45 46 47 48 49 50 51 52 53 61 62 63 64 65 66
      67 68 69 70 71 72 73
```

Cards are created in more detail this time as I have to note different suits and numbers to check for cards in a row in a certain suit. So first suite (ace,2-10,jack,queen,king=ι13). Subsequent suits are increased by 20 so each card has a unique number and each suit has each card increasing by 1. Note there is a gap between each suit(14-20, 34-40 and 54-60).
First we also need a fns to sort the drawn cards in order:

```
     sort←{ω[⍋ω]}
```

Now lets look for runs of 3 or more (ie 2 differences=1 for a run of 3, 3 diffs=1 for a run of 4 and 4 diffs=1 for a run of 5)

```
 2≤+/⎕←1=⎕←|2-/⎕←sort cards[5?52]
5 10 26 47 50   ⍝ shows(⎕) 5 randomly selected cards sorted
5 16 21 3       ⍝ shows(⎕) the 4 diffs between pairs of above cards(2-/)
0 0 0 0         ⍝ shows that none of differences =1
0               ⍝ shows that no sequence was longer than 2
```

Now lets take the shows(⎕) out and run it a million times

```
 {(+/{2≤+/1=|2-/sort cards[5?52]}¨ωρ0)÷ω}1000000
0.037195   ⍝ ~3.7% of time will I get a run of 3 or more in the same suit.
```

Now lets look at runs for 3, 4 and 5 separately

```
 {(+/{2=+/1=|2-/sort cards[5?52]}¨ωρ0)÷ω}1000000
0.035668         ⍝ ~3.6% runs of 3
 {(+/{3=+/1=|2-/sort cards[5?52]}¨ωρ0)÷ω}1000000
0.001466         ⍝ ~.1% runs of 4
 {(+/{4=+/1=|2-/sort cards[5?52]}¨ωρ0)÷ω}1000000
0.000013         ⍝~.0013% runs of 5
```

3. What are odds of something simple like 1 pair? This probability is 0.422569

How would you go about this? (Hint: make each suit string equal)
http://www.math.hawaii.edu/~ramsey/Probability/PokerHands.html

An Optimal Stopping Problem: Dating For Dummies ****

How many should you date before deciding to marry next one better than anyone you dated so far if you want best odds of getting best 1 or maybe 1 in top 10? Assume **nd**=# of total people you could date, **s**=# you date and **top**=# of best people you would be willing to accept(1 if you want best, 2 if either of top 2 would be good enough etc.)

Here is the fns: The actual code is boldface. All the rest is comments.

```
dates←{  ⍝ each time fns called returns 1 if found good enough mate else 0
⍝ Chapter 20:An Optimal Stopping Problem or maybe Dating for Dummies
⍝ How many to date before picking a mate from book by Paul Nahin 2008
⍝ Digital Dice:Computational Solutions to Practical Probability Problems
⍝ or https://www.ted.com/talks/hannah_fry_the_mathematics_of_love#t-598603
⍝ Input and Output
⍝ return 1 if pick person in "top" range of sample "s" by picking first
⍝ date who is better than the best of "nd" people in the dated group.
     nd←⍺ ⋄ s top←⍵ ⍝ nd=# dates s=sample size top=# of good enough dates
     (s=0)∨(s=nd):top≥?s ⍝ if picked first or last date odds are: ~top/s
     ranks←s?s ⍝ make random ranks for all dates. (1=best to s=worst)
     bestdate←⌊/nd↑ranks ⍝ bestdate=lowest rank(nd↑ranks) of those dated
     left←nd↓ranks ⍝ left=rest taking away those dated at beginning.
     better←(left<bestdate)/left ⍝ better=1 or 0 for  each left<bestdate
     top≥1↑better,10000000 ⍝ 1 if 1st pick of better in top range else 0
⍝ sample probability runs:(only repeated run averages are really useful)
⍝ 2 dates 11 1 ⍝ nd=2 s=11 top=1, return 1 if best=(next date>first two)
⍝ following all call fns 10,000 times & average to get odds of success
⍝ next 2 from book show odds for 0 to 11 dates from total of 11 people
⍝ x,[1.5] {4⍕⊃avg ⍵ dates¨⊂11 1}¨10000⍴¨x←0,⍳10 ⍝ p94 table probabilities
⍝ x,[1.5] {4⍕⊃avg ⍵ dates¨⊂11 3}¨10000⍴¨x←0,⍳10 ⍝ odds if ok with top 3
⍝ next example s=1000, you date 7 various #'s (50×⍳7)[50 100 150...350]
⍝ x,[1.5] {4⍕⊃avg ⍵ dates¨⊂1000 20}¨10000⍴¨x←50×⍳7 ⍝ odds mate in top 20
}
```

On **page 94** of Paul Nahin's book there's a **probability table** that the above program will approximate. So let's run it 10,000 times for each possible number of dates and average results to get his table for each number of possible dates. So if the number of all possible dates is **nd**=11 & you want the very best person(**top**=1). What are the odds of you getting the best person if you date 0,1,2,3,4,5,6,7,8,9, or 10 people before picking.

```
     x,[1.5] {4⍕⊃avg ⍵ dates¨⊂11 1}¨10000⍴¨x←0,⍳10  ⍝ avgs 100,000 trials
 0  0.0913    ⍝ actual odds first person is best 1/s = 1/11 =.0909  9%
 1  0.2649
 2  0.3448    ⍝ odds improving but still better to keep dating
 3  0.3959
 4  0.3991    ⍝ best odds ~40% if date 4 then pick next 1 better than 1-4
 5  0.3777    ⍝ odds begin to decline ~38%. You should have picked sooner.
 6  0.3541    ⍝ 1/e=
 7  0.2994
 8  0.2456    ⍝ <25% chance of finding best one
 9  0.1705
10  0.0906    ⍝ actual odds last person is best 1/s = 1/11 =.0909  9%
```

So if 11 people to date best odds of finding best 1 is date 4 then pick next one better than any of first 4. But remember this is only best odds ~40%. ~60% of time you will miss very best one. Experiment seeing odds of getting 1 of the top 2 or 3. Or imagine 1000 in dating pool. How many should you date to get maybe 1 of the top 20. Running this program may not be quite as much fun as dating but it's lots faster and bit cheaper than having a couple hundred dates. Many decisions can be improved using this method. Can you think of some? How about: finding/buying/selling/renting: career, school, pet, house, apartment, car, bike. Anything that's gone once you say no. Or maybe you figure you want to have children by age 35 and you are now say 18. How many years should you date before you pick the next one who is better than any you have dated so far. Here is the answer:

```
     x,[1.5] {4⍕⊃avg ⍵ dates¨⊂17 1}¨100000⍴¨x←0,⍳17  ⍝ avgs 100,000 trials
 0  0.0587  ⍝ actual odds that first year is best 1/s = 1÷17 =.0588  5.88%
 1  0.2005
 2  0.2803
 3  0.3300
 4  0.3634
 5  0.3785
 6  0.3854 ⍝ best odds ~38% so date 6yrs then pick next 1 better than 1-6
 7  0.3824 ⍝ (for this simple case of picking the very best one there is
 8  0.3701    easier way to calculate based on e [the base of the natural
 9  0.3483    logarithms e=2.71828 or *1 in APL] simply do n×1÷e or in APL
10  0.3230    17×1÷*1 = 17×0.3678794411714423 = 6.25395049991452 so
11  0.2909    best odds is about 36.79% of the way(roughly 1/3 of the 17
12  0.2539    years or about 6 years it’s time to pick your partner)
13  0.2125
14  0.1637 ⍝ odds declining. You should have picked sooner.
15  0.1130
16  0.0576
17  0.0582 ⍝ actual odds that last year is best 1/s = 1÷17 =.0588  5.88%
```

The twins problem (using math, Matlab and APL) ***

From: **Will You Be Alive 10 Years From Now? by Paul Nahin 2014**
A Very Fun Book of curious questions in probability

In February 2008 I received a very interesting e-mail from Bruce C. Taylor, a professor of biomedical engineering at the University of Akron. Bruce had just been reading my book, *Duelling Idiots* (Princeton 2002), and that prompted him to write to me. Here's what Bruce wrote:

> I have an interesting probability problem that I have not been able to solve and I am just curious to see if you can come up with a solution. The problem came up when in one of our classes here I was assigning lab groups using a random number generator. As it turns out the class had 20 students, two of whom were related (twin sisters). Well, as luck would have it, the two sisters ended up in the same lab group of four. I had divided the class into five groups of four students. I, and a colleague, got to wondering what was the probability that the two sisters would end up in the same group. I originally thought that this would be a trivial problem but so far it has beaten me. I did write a MATLAB? program to solve the problem via a probabilistic model and I came up with a probability of 0.16 after 100,000 repetitions. I think that this is the correct answer but I can't, for the life of me, arrive anywhere near the same answer analytically. I thought maybe you'd like to take a crack at it.

Well, who could resist that?

> After a bit of thought I did arrive at a theoretical result, a rational fraction approximately equal to 0.1579, and so I wrote back to Bruce to ask, "You said the [Monte Carlo] estimate was 0.16. Was it actually somewhat less?" Back came Bruce's response: "I ran the simulation three times at 100,000 reps. each and came up with the following: (1) 0.1591, (2) 0.1570, (3) 0.1557." Not too bad an agreement with my fraction. I then wrote my own MATLAB? simulation code, ran it for ten million repetitions, and got an estimate of 0.1579092, an even better agreement with my theoretical fraction.

2.2 THEORETICAL ANALYSIS

To theoretically derive the answer to Bruce's question, here's what I sent him,, where (${}^{X}{}_{y}$) is, as in the first problem, the binomial coefficient x!/(x — *y)!y! ,* with x and y both non-negative integers and y≤ x.

First, to find the total number of ways (TNW) to randomly place 20 students into 5 groups of 4 each, imagine 5 bins. In the first bin we place 4 from 20, then 4 from the remaining 16 in the second bin, then 4 from remaining 12 in third bin, and so on. Combination formula follows

Thus, $\mathrm{TNW}=\binom{20}{4}\binom{16}{4}\binom{12}{4}\binom{8}{4}\binom{4}{4}$

```
×/4!20 16 12 8 4 ⍝ in APL 4 paired each # right of comb symbol ! then ×/ multiplies
```

Next, to find the total number of ways that the twins are together (TNWTT) in the same bin, we first imagine that the twins are glued together. When we select a twin, we automatically select the other one, too. There are 5 ways to place the glued twins into one of the bins, leaving 18 students. There are (${}^{1}{}_{2}{}^{8}$) ways to select the 2 students who join the twins, leaving 16 students. We then finish the analysis as before, that is

$$\mathrm{TNWTT}=5\ \binom{18}{2}\binom{16}{4}\binom{12}{4}\binom{8}{4}\binom{4}{4}.$$

```
      5××/2 4 4 4 4!18 16 12 8 4 ⍝ in APL
```

Note: ! is combination symbol **×/** multiplies all combinations **5×** multiplies that result by 5

Now the probability we are after is:

$$\frac{\mathrm{TNWTT}}{\mathrm{TNW}} = \frac{5\binom{18}{2}}{\binom{20}{4}} = \frac{5\,18!/16!2!}{20!/16!4!} = 5\frac{18!4!}{20!2!} = 5\frac{(4)(3)}{(20)(19)}$$

= 3/19 = .15789….

```
(5×2!18)÷(4!20) = 0.15789473684210525 ⍝ Using APL combination symbol !
```

Note: !6 is factorial 6 & 2!6 is combinations of 6 taken 2 at a time.

Now, as easy as the above analysis may appear, an early reviewer of this book (Nick Hobson) pointed out to me that there is an even easier way to see the result in a flash. A total of 20 lab slots are to be filled, with 4 slots in each lab section. One of the twins, of course, has to be in *some* lab section, leaving 3 slots in *that* section still available out of the 19 total slots that are still available. So, the probability that our second twin gets one of those 3 slots (and so joins her sister) is 3/19. That's it!

2.3 COMPUTER SIMULATION

To write a Monte Carlo simulation, I found the following imagery helpful. (I wrote my simulation code before receiving Nick's clever observation, so perhaps there is a better way to simulate—I'll leave that for *you* to explore!) I stalled by visualizing the 20 students lined up in front of me in some (random) order, standing in a row, shoulder to shoulder. Each holds a slip of paper. These slips each have a single number on them; there's a 2 on each twin's slip, while all the other students have a 1 on their slips. Starting at the far left (student 1), the first four students are assigned to lab section 1, the next four students to lab section 2, and so on, with students 17 through 20 assigned to lab section 5. To simulate the placement of the twins into their lab sections, all we need do is randomly generate two different integers from 1 to 20, integers that determine the positions where the twins stand in the shoulder-to-shoulder row.

The simulation code can determine if the two twins have been assigned to the same lab section by simply adding up the numbers, in each lab section, on the paper slips held by the students in that section. If a lab section has neither twin, the group sum will be 4, while if a lab section has one twin, the group sum will be 5. A group sum of 6, however, means we have a lab section that contains both twins. This is the decision logic behind the simulation code **twins.m. I** make no claims that **twins.m** is a superoptimal (in some sense) code, just that it is easily understood and executes in a reasonably short time (ten million repetitions on my quite ordinary, bottom-of-the-line computer required less than 23 seconds to run). After the code listing, I'll give you a quick walkthrough of what each line is doing (the line numbers at the far left are not part of the code but are included simply as reference tags for the walkthrough).

twins.m

```
01 together=0;
02 for loop1=1:10000000
03      lab=ones(1,20);
04      twin1=floor(20*rand)+1;
05      twin2=twin1;
06      while twin 1= =twin2
07              twin2=floor(20"rand)+1;
08      end
09      lab(twin1)=2;
10      lab(twin2)=2;
11      groupsum=zeros(1,5);
12      for loop2=1:5
```

```
13          x=4*(loop2-1);
14          for loop3=1:4
15               groupsum(loop2)=groupsum(loop2)+lab(x+loop3);
16          end
17     end
18     for loop4=1:5
19          if groupsum(loop4)= =6
20               together=together+1;
21          end
22     end
23 end
24 together/10000000
```

Line 01 initializes the variable *together* to zero; at the end of ten million simulations *together* will be the number of simulations in which the twins were assigned to the same lab section. Lines 02 and 23 define the outer *for/end* loop that cycles the code through the ten million simulations. Line 03 defines the row vector *lab,* with all of its 20 elements initially equal to 1. The value *lab(k)* is the number written on the slip of paper held by the student in the kth row position. Initially, then, all 20 students have a 1 on their individual slips of paper. Line 04 assigns *twin1* equal to an integer value selected at random from 1 to 20, and line 05 assigns the same integer to *twin2.* Since the two twins can't, of course, have the same position in *lab,* lines 06 through 08 then continually assign *twin2* a new random integer value until *twin1* and *twin2* have different integer values. Lines 09 and 10 write a 2 on the slip of paper each twin holds, leaving the other 18 students holding slips of paper each with a 1. Line 11 initializes all five elements of the row vector *group-sum* to zero. The two nested loops defined by lines 12 through 17 run through the 20 elements of *lab,* four at a time, from left to right, and generate the five element values of *groupsum.* Finally, the two nested loops defined by •lines 18 through 22 check each element of *groupsum* and, if a value of 6 is detected (indicating both twins are in the same section), then *together* is incremented by one. Once the ten million simulations are finished, line 24 prints the code's estimate of the probability of the twins being in the same lab section (0.1579092), an estimate very close to the theoretical value.

Now My 1 line of APL to compare to Nahim’s 24 lines of Matlab

```
      5×avg{1 1≡1 2∊4?⍵}¨10000000⍴20
0.157496
```

Let me explain the code. Apl works from right to left **10000000⍴20** creates 10 million 20’s. The each symbol ¨ calls the unnamed program between the **{}** 10 million times passing it one 20 each time assigning the 20 to the symbol ⍵. **4?20** finds 4 different random numbers between 1 and 20. Then the **1 2∊** sees if each of the numbers 1 & 2 is a member of the set of 4 random numbers. If it is it returns a 1 otherwise it returns a 0. I have chosen 1 and 2 as the id numbers for the twins so if there is a 1 and a 2 in the 4 numbers it means the twins are together in the first group. If it returns a 1 0 or 0 1 it means only one of the twins was in the group. If 0 0 it means neither of the twins was in the group. Finally match ≡ compares the two numbers to see if they match it’s left argument of **1 1**. If they match a 1 is returned otherwise a 0 is returned. So after the program inside the **{}** runs 10 million times we have a string of 1’s and 0’s which are averaged by the **avg** program to see the proportion of times the twins are both in the first group. If we had looked at 5 groups of 4 people each time we would have found 5 times more matches so I multiplied this number by 5 to get the expected percentage of times the twins would have been in one of the 5 groups.

As you can see I cheated a little as the above example only looks at 1 of the 5 groups and then multiplies the average by 5 to get Monte Carlo estimate. So I am really doing 5 times less computation. If I change to 50 million instead of 10 million I get a workspace full error on my computer. The APL program does the data as a vector instead of looping around and around as Matlab does and thus requires all the memory at one time. So to be fair I did 10 million runs 5 times to get my 50 million here which is equivalent to the 10 million Matlab example. So here it is:

```
     5×avg{avg{1 1≡1 2∊4?⍵}¨10000000⍴⍵}¨5⍴20
0.1579035
```

I used this to compute the average **avg←(+⌿ ÷ ≢)** . For example: **avg 4 5 6** is sum of numbers (⍵=4 5 6 and +⌿⍵=18) divided ÷ by number of numbers (w=4 5 6 and **≢ ⍵=3**), which is simply the sum of the numbers **+⌿** divided ÷ by number of numbers **≢**. Thus 18÷3=6 the average.

Here is another run with the apl program to compute the average included in the one line APL program. It also shows that 50 million runs is probably enough to get a pretty good estimates of the theoretical number of .1579. Try APL yourself my website **jerrymbrennan.com**

```
      avg←(+⌿ ÷ ≢) ⋄ 5×avg{avg{1 1≡1 2∊4?⍵}¨10000000⍴⍵}¨5⍴20
 0.1579821
```

With APL there are a number of somewhat similar ways to compute this percentage. Below are 4 different ways compared to see which is fastest using a builtin timer program **]runtime** with 4 input strings of the 4 different methods. It looks like the above method tested first below using membership ∊ is not the fastest though the fourth method using union ∩ only takes 5% less time. Reduction ∧/ and Plus Reduction +/ both seem to take a bit longer.

Now below is a long one line APL call to util program]runtime passing it the 4 method & below that are 4 result times compared:

```
]runtime '5×avg{1 1≡1 2∊4?⍵}¨100000⍴20' '5×avg{∧/1 2∊4?⍵}¨100000⍴20'
'5×avg{2=+/1 2∊4?⍵}¨100000⍴20' '5×avg{1 2≡1 2∩4?⍵}¨100000⍴20' -compare
```

```
  5×avg{1 1≡1 2∊4?⍵}¨100000⍴20 → 3.4E¯1 |   0% □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
* 5×avg{∧/1 2∊4?⍵}¨100000⍴20   → 3.9E¯1 | +12% □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
* 5×avg{2=+/1 2∊4?⍵}¨100000⍴20 → 3.7E¯1 |  +6% □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
* 5×avg{1 2≡1 2∩4?⍵}¨100000⍴20 → 3.3E¯1 |  -5% □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

Now go to **JMB.APLCloud.com** where you can try all the APL examples or any other APL you want. There’s also my 60 page pdf manual with many more examples & many online tutorial, videos teaching APL, complete interactive statistics package, links to getting educational APL free and 800 page free download pdf APL manual. If questions please email me Jerry M Brennan PhD at jbrennan@hawaii.rr.com or go to my website at jerrymbrennan.com

Generate Numbers 1-10 From Digits 1-4 Using APL Symbols ****

Your assignment is to find APL symbols that operate on vector: a←⍳4 and find a set of symbols that will generate each of the numbers 1-10 with the fewest characters. For example: a[1] or 1⊃a would both work to generate 1. The second one is preferred as it uses less characters(3 instead of 4).

HERE IS A PROGRAM I WROTE TO SCORE YOUR RESULTS.

```
  ScoreNumbers←{ ⍝ ⍵ rt arg is your trys ie '1↑a' '-/⌽a' etc
  ⍺←(⍳10)(⍳4) ⍝ default left arg is answers & #'s to use
  ans←1⊃⍺ ⋄ a←2⊃⍺ ⍝ answers & "a" values to use to get answers
  avg←{(+/⍵)÷⍴⍵} ⍝ define average fns
  try←,¨(⍴ans)↑⍵,500⍴⊂'¯99' ⍝ expand your trys to = the length of ans
  try←(¯1+try⍳¨'⍝')↑¨try  ⍝ elim comments on lines
  r←⊂'1=right 0=wrong: ',⍕score←⊃¨ans=⍎¨try
  r,←⊂'Lengths of each: ',⍕⊃¨⍴¨try
  r,←⊂'# and % correct: ',(⍕n),7 2⍕100×(n←+/score)÷⍴score
  r,←⊂'Correct avg len: ',⍕avg⊃¨⍴¨score/try
  ↑r
  ⍝ ans for: (⍳20)(⍳4) ScoreNumbers one2four [#’s 1-20 using 1-4]
  ⍝ ans for: (0,⍳20)(4⍴4) ScoreNumbers fourfour [#’s 0-20 using 4 4 4 4]
 }
```

So if you had 3 answers done you could score it like this:

```
      Mytries←'1↑a' '-/⌽a' 'a[3]'
      ScoreNumbers Mytries
1=right 0=wrong: 1 1 1 0 0 0 0 0 0 0
Lengths of each: 3 4 4 1 1 1 1 1 1 1
# and % correct: 3 30
Correct avg len: 3.666666667
```

EXTRA CREDIT 1: Find the numbers 1-20. Change ScoreNumbers default left argument **in** line1 to (⍳20)(⍳4) **like this:** (⍳20)(⍳4) ScoreNumbers Mytries **Note also the 0’s to fill unknowns if you are not sure of some of them.**

```
      Mytries←'1↑a' '-/⌽a' 'a[3]' '0' '0' '0' '0' '0' '0' '0' '0'  '×/2↓a'
      (⍳20)(⍳4) ScoreNumbers Mytries
1=right 0=wrong: 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
Lengths of each: 3 4 4 0 0 0 0 0 0 0 5 3 3 3 3 3 3 3
# and % correct: 4 20
 3  4  4  5
Correct avg len: 4 ⍝ my correct solution for ⍳20 was 7.7. can you beat it?
```

EXTRA CREDIT 2: Use as you input 4 4’s & find numbers 0-20. You must use all 4 4’s to get each number. Here’s my answers(hidden in variable X). Can you beat it?

```
(0,⍳20)(4⍴4) ScoreNumbers X
1=right 0=wrong: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Length of each:11 11 13 11 15 16 14 13 6 13 17 13 13 12 11 13 3 13 15 24 13
# and % correct: 21 100
11 11 13 11 15 16 14 13 6 13 17 13 13 12 11 13 3 13 15 24 13
Correct avg len: 12.85714286
```

GENERATE NUMBERS - SOME ANSWERS FOR: 1-20 USING ι4 AND 4⍴4

Possible answers for : First find #’s 1-20 using a←ι4 (1 2 3 4)

```
(ι20)(ι4) ScoreNumbers one2four ⍝ use some or all digits repeats allowed

1↑a                        ⍝  1
2⊃a                        ⍝  2
3⊃a                        ⍝  3
3↓a                        ⍝  4
+/a[1 4]                   ⍝  5
!/a                        ⍝  6
+/2↓a                      ⍝  7
×/a[2 4]                   ⍝  8
*/1↓⌽a                     ⍝  9
+/a                        ⍝ 10
(×/a[3 4])-1⊃a             ⍝ 11
×/a[3 4]                   ⍝ 12
a[1]+×/a[3 4]              ⍝ 13
a[2]×+/2↓a                 ⍝ 14
a[3]+×/2↓a                 ⍝ 15
(4⊃a)*2                    ⍝ 16
a[1]+(4⊃a)*2               ⍝ 17
a[3]××/a[2 3]              ⍝ 18
(*/a)+a[2]××/1↓⌽a          ⍝ 19
+/2/a                      ⍝ 20
Lengths of each: 3 3 3 3 8 3 5 8 6 3 14 8 13 10 10 7 12 13 17 5 avg=7.7
```

Possible answers for : Now find #’s 0-20 using a←4⍴4 (4 4 4 4)

```
(0,ι20)(4⍴4) ScoreNumbers fourfour    ⍝ note: You must use every 4 once.

+/(2↑a)-2↓a                ⍝  0
×/(2↑a)÷2↓a                ⍝  1
(÷/2↑a)+÷/2↓a              ⍝  2
(+/3↑a)÷3↓a                ⍝  3
a[1]+a[2]×-/2↓a            ⍝  4
(a[3]+×/2↑a)÷3↓a           ⍝  5
(+/!2↑a)÷+/2↓a             ⍝  6
(+/2↑a)-÷/2↓a              ⍝  7
-/⌽+\a                     ⍝  8
(÷/2↑a)++/2↓a              ⍝  9
+/a[1]+a[2 3]÷4⊃a          ⍝ 10
(+/3↑a)-⌊⍟3↓a              ⍝ 11
(×/2↑a)-⌊/2↓a              ⍝ 12
(+/3↑a)+⌊⍟4                ⍝ 13
(+/3↑a)+⌈⍟4                ⍝ 14
(×/2↑a)-÷/2↓a              ⍝ 15
+/a                        ⍝ 16
(×/2↑a)+÷/2↓a              ⍝ 17
(×/2↑a)++/⌊⍟2↓a            ⍝ 18
(×/a[2 3])+(⌈⍟1↑a)+⌊⍟3↓a   ⍝ 19
(×/2↑a)+⌊/2↓a              ⍝ 20
Lens:11 11 13 11 15 16 14 13 6 13 17 13 13 12 11 13 3 13 15 24 13 avg=12.8
```

WORKING WITH TABLES **

Company wants to compare actual & forecasts for 4 products for 6 months.

```
     Forecast←4 6⍴150 200 100 80 80 80 300 330 360 400 500 520 100 250 350
 380 400 450 50 120 220 300 320 350  ⍝ Forecast reshape(⍴) to 4x6 table

     Actual←4 6⍴141 188 111 87 82 74 321 306 352 403 497 507 118 283 397
 424 411 409 43 91 187 306 318 363   ⍝ Actual reshape(⍴) to 4x6 table
```

```
      Forecast
150 200 100  80  80  80
300 330 360 400 500 520
100 250 350 380 400 450
 50 120 220 300 320 350

      Actual
141 188 111  87  82  74
321 306 352 403 497 507
118 283 397 424 411 409
 43  91 187 306 318 363
```

```
    Forecast-Actual
  9  12 ¯11  ¯7  ¯2   6
¯21  24   8  ¯3   3  13
¯18 ¯33 ¯47 ¯44 ¯11  41
  7  29  33  ¯6   2 ¯13
```

```
      Forecast,¨Actual
 150 141  200 188  100 111  80 87    80 82    80 74
 300 321  330 306  360 352  400 403  500 497  520 507
 100 118  250 283  350 397  380 424  400 411  450 409
 50 43    120 91   220 187  300 306  320 318  350 363
```

```
      +fa←(⊂4 0)⍕¨Forecast,¨Actual ⍝ each col is 4 wide with 0 decimals
  150 141   200 188   100 111    80  87    80  82    80  74
  300 321   330 306   360 352   400 403   500 497   520 507
  100 118   250 283   350 397   380 424   400 411   450 409
   50  43   120  91   220 187   300 306   320 318   350 363
```

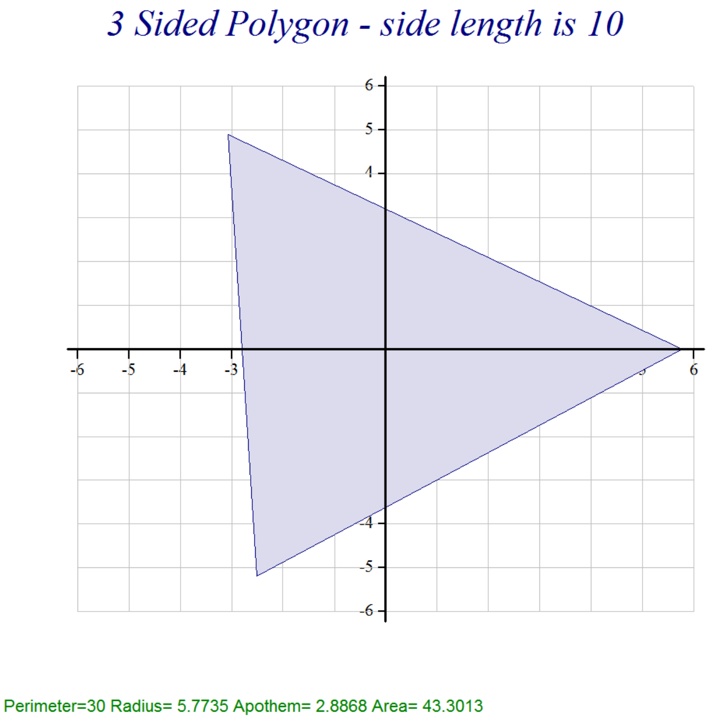
```
      (⊂4 0)⍕¨Forecast,¨Actual,¨Forecast-Actual
 150 141   9   200 188  12   100 111 ¯11    80  87  ¯7    80  82  ¯2    80  74   6
 300 321 ¯21   330 306  24   360 352   8   400 403  ¯3   500 497   3   520 507  13
 100 118 ¯18   250 283 ¯33   350 397 ¯47   380 424 ¯44   400 411 ¯11   450 409  41
  50  43   7   120  91  29   220 187  33   300 306  ¯6   320 318   2   350 363 ¯13
```

```
     ((⊂'Prod\Month'),⍕¨⍳1↑⍴fa),((⍕¨⍳1↓⍴fa),¨6⍴⊂':Fo Act')⍪fa ⍝ label rows & cols
Prod\Month  1:Fo Act  2:Fo Act  3:Fo Act  4:Fo Act  5:Fo Act  6:Fo Act
1            150 141   200 188   100 111    80  87    80  82    80  74
2            300 321   330 306   360 352   400 403   500 497   520 507
3            100 118   250 283   350 397   380 424   400 411   450 409
4             50  43   120  91   220 187   300 306   320 318   350 363
```

Plotting Regular Polygons **

```
R←PolyPlot(n s);y;x;y;foot;range;x0;y0;Deg2Rad;theta;i;radius;py;px;pct;area;apothem
 ⍝ n is # sides s=side length. So: PolyPlot 5 10 plots 5 sided polygon with each side=10
 Deg2Rad←{⍵×○1÷180}  ⍝ fns to convert degrees to radians for input to trigonometric fns

 radius←s÷2×1○Deg2Rad 180÷n            ⍝ center to a vertex    1○ is sine
 apothem←s÷2×3○Deg2Rad 180÷n           ⍝ center to midpt side  3○ is tangent
 area←(n×s*2)÷4×3○Deg2Rad 180÷n        ⍝ area of polygon       3○ is tangent

 x0←y0←0 ⍝ x y location of center of polygon on plot
 ⍝ see http://www.mathopenref.com/polygonregulararea.html for following formulas
 theta←(360÷n)×i←0,(⍳n-1),0 ⍝ theta is angle with the x axis plot based on # of sides (n)
 x←x0+radius×2○Deg2Rad theta+i×(2×○1)÷n ⍝ x vertice locations   2○ is cosine
 y←y0+radius×1○Deg2Rad theta+i×(2×○1)÷n ⍝ y vertice locations   1○ is sine

 ch.New 350 350  ⍝ trying to make x y lengths the same but failing
 ch.Set'Head'((⍕n),' Sided Polygon - side length is ',⍕s)
 ch.Set'Footer'(('Perimeter=',⍕n×s),(' Radius=',⍕4⍕radius),(' Apothem=',⍕4⍕apothem),('
Area=',⍕4⍕area))
 ch.Set¨(⊂¨'Xrange' 'Yrange'),¨range←⊂¯1 1×⌈/|x,y
 ch.Set¨('Xint' 0)('Yint' 0)('forcezero')('XYPLOT,GRID')
 ch.Set'style' 'surface'
 ch.Plot⍉↑x y
 PG←ch.Close
R←'View PG ⍝ to see it'
```

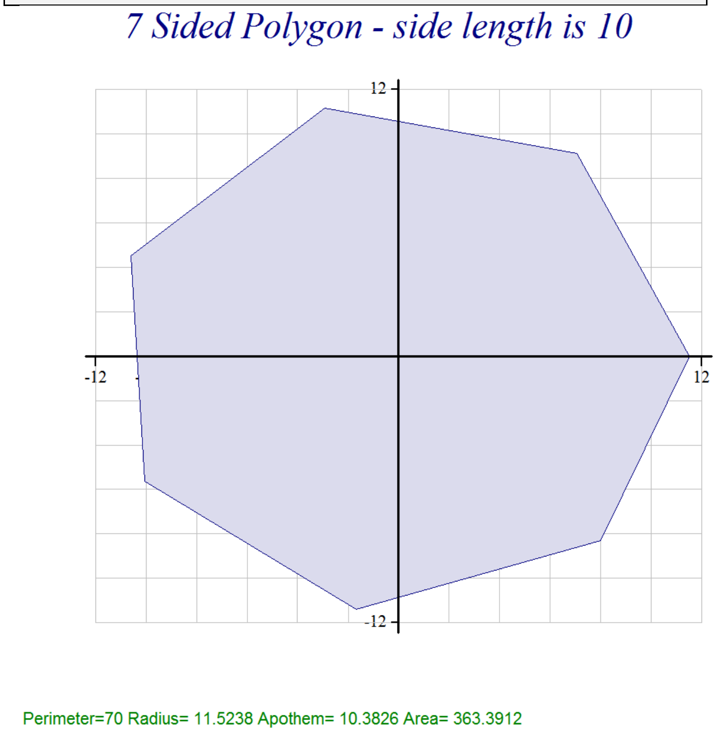
```
PolyPlot 3 10
View PG ⍝ to see it
```



3 Sided Polygon - side length is 10

Perimeter=30 Radius= 5.7735 Apothem= 2.8868 Area= 43.3013

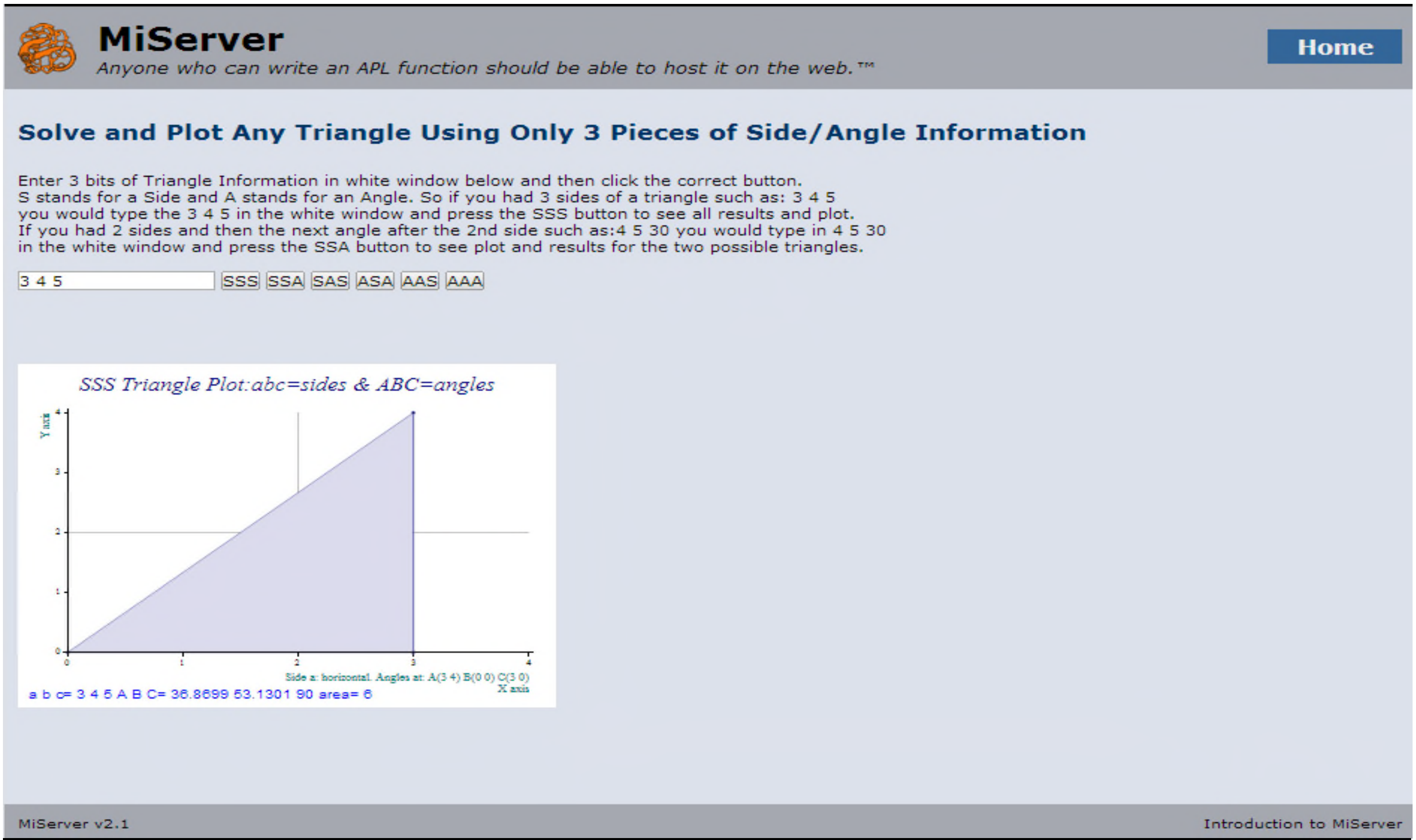
```
PolyPlot 7 10
View PG ⍝ to see it
```



7 Sided Polygon - side length is 10

Perimeter=70 Radius= 11.5238 Apothem= 10.3826 Area= 363.3912

Plotting Any Triangle Given Some Sides & Angles **



MiServer

Anyone who can write an APL function should be able to host it on the web.™

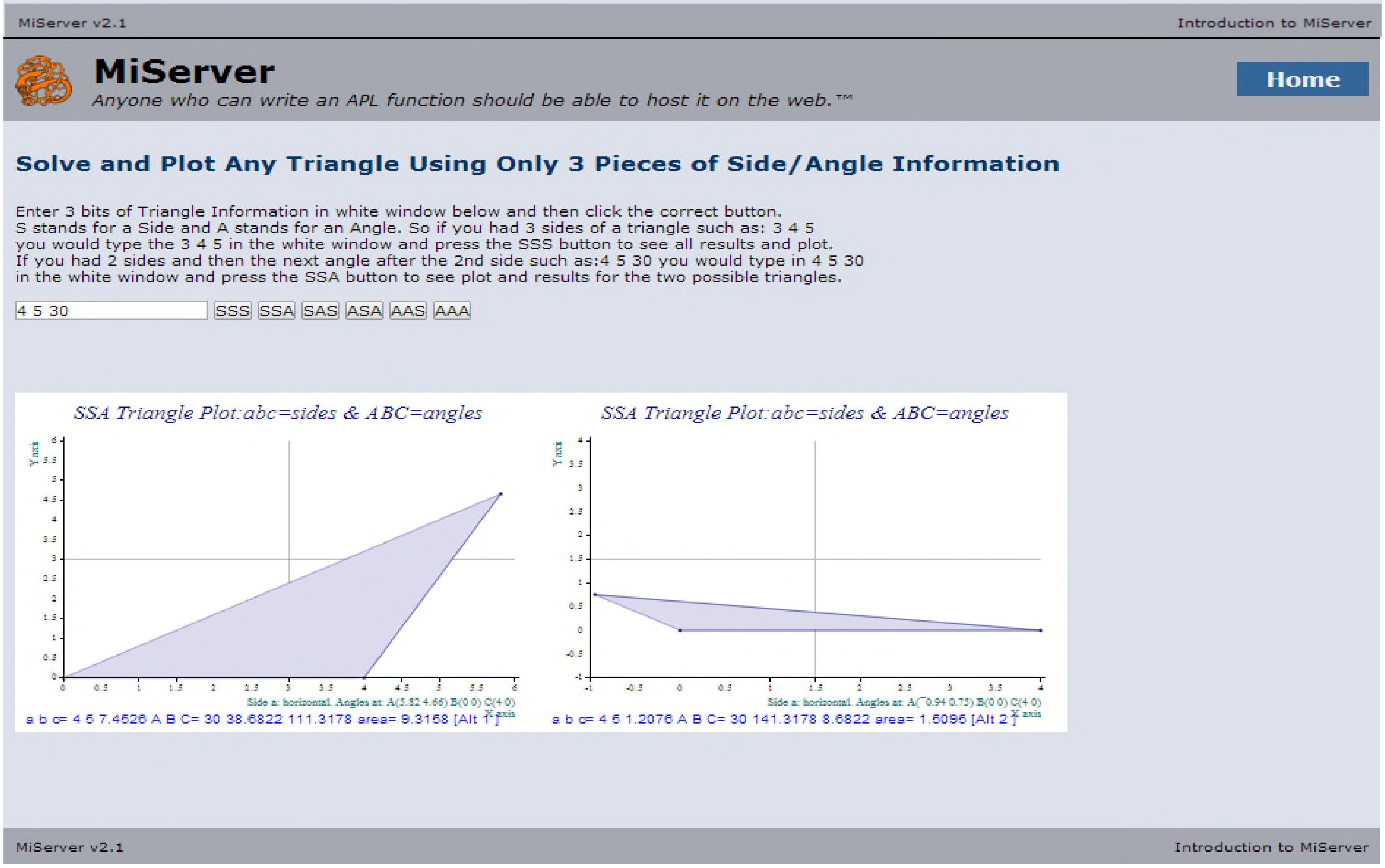
Solve and Plot Any Triangle Using Only 3 Pieces of Side/Angle Information

Enter 3 bits of Triangle Information in white window below and then click the correct button.
S stands for a Side and A stands for an Angle. So if you had 3 sides of a triangle such as: 3 4 5
you would type the 3 4 5 in the white window and press the SSS button to see all results and plot.
If you had 2 sides and then the next angle after the 2nd side such as:4 5 30 you would type in 4 5 30
in the white window and press the SSA button to see plot and results for the two possible triangles.

3 4 5

SSS
SSA
SAS
ASA
AAS
AAA
SSS Triangle Plot:abc=sides & ABC=angles
Y axis
X axis
Side a: horizontal. Angles at: A(3 4) B(0 0) C(3 0)

a b c= 3 4 5 A B C= 36.8699 53.1301 90 area= 6



MiServer

Anyone who can write an APL function should be able to host it on the web.™

Solve and Plot Any Triangle Using Only 3 Pieces of Side/Angle Information

Enter 3 bits of Triangle Information in white window below and then click the correct button.
S stands for a Side and A stands for an Angle. So if you had 3 sides of a triangle such as: 3 4 5
you would type the 3 4 5 in the white window and press the SSS button to see all results and plot.
If you had 2 sides and then the next angle after the 2nd side such as:4 5 30 you would type in 4 5 30
in the white window and press the SSA button to see plot and results for the two possible triangles.

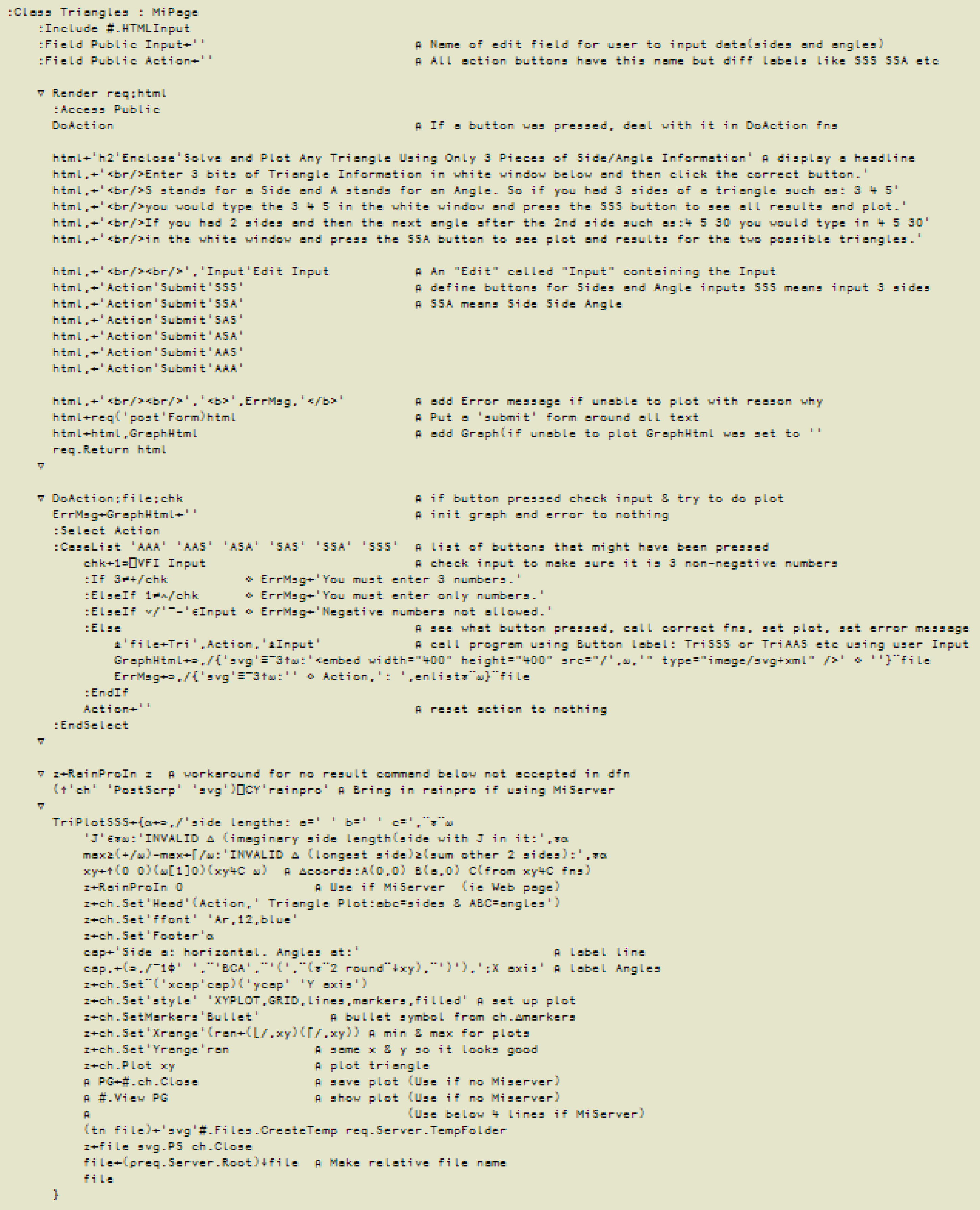
4 5 30

SSS
SSA
SAS
ASA
AAS
AAA
SSA Triangle Plot:abc=sides & ABC=angles
Y axis
X axis
Side a: horizontal. Angles at: A(5.82 4.66) B(0 0) C(4 0)

a b c= 4 5 7.4526 A B C= 30 38.6822 111.3178 area= 9.3158 [Alt 1]

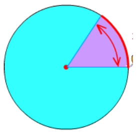
SSA Triangle Plot:abc=sides & ABC=angles
Y axis
X axis
Side a: horizontal. Angles at: A(¯0.94 0.75) B(0 0) C(4 0)

a b c= 4 5 1.2076 A B C= 30 141.3178 8.6822 area= 1.5095 [Alt 2]



```
:Class Triangles : MiPage
    :Include #.HTMLInput
    :Field Public Input←''                           ⍝ Name of edit field for user to input data(sides and angles)
    :Field Public Action←''                          ⍝ All action buttons have this name but diff labels like SSS SSA etc

    ∇ Render req;html
      :Access Public
      DoAction                                       ⍝ If a button was pressed, deal with it in DoAction fns

      html←'h2'Enclose'Solve and Plot Any Triangle Using Only 3 Pieces of Side/Angle Information' ⍝ display a headline
      html,←'<br/>Enter 3 bits of Triangle Information in white window below and then click the correct button.'
      html,←'<br/>S stands for a Side and A stands for an Angle. So if you had 3 sides of a triangle such as: 3 4 5'
      html,←'<br/>you would type the 3 4 5 in the white window and press the SSS button to see all results and plot.'
      html,←'<br/>If you had 2 sides and then the next angle after the 2nd side such as:4 5 30 you would type in 4 5 30'
      html,←'<br/>in the white window and press the SSA button to see plot and results for the two possible triangles.'

      html,←'<br/><br/>','Input'Edit Input           ⍝ An "Edit" called "Input" containing the Input
      html,←'Action'Submit'SSS'                      ⍝ define buttons for Sides and Angle inputs SSS means input 3 sides
      html,←'Action'Submit'SSA'                      ⍝ SSA means Side Side Angle
      html,←'Action'Submit'SAS'
      html,←'Action'Submit'ASA'
      html,←'Action'Submit'AAS'
      html,←'Action'Submit'AAA'

      html,←'<br/><br/>','<b>',ErrMsg,'</b>'         ⍝ add Error message if unable to plot with reason why
      html←req('post'Form)html                       ⍝ Put a 'submit' form around all text
      html←html,GraphHtml                            ⍝ add Graph(if unable to plot GraphHtml was set to ''
      req.Return html
    ∇

    ∇ DoAction;file;chk                              ⍝ if button pressed check input & try to do plot
      ErrMsg←GraphHtml←''                            ⍝ init graph and error to nothing
      :Select Action
      :CaseList 'AAA' 'AAS' 'ASA' 'SAS' 'SSA' 'SSS'  ⍝ list of buttons that might have been pressed
          chk←1=⎕VFI Input                           ⍝ check input to make sure it is 3 non-negative numbers
          :If 3≠+/chk          ⋄ ErrMsg←'You must enter 3 numbers.'
          :ElseIf 1≠∧/chk      ⋄ ErrMsg←'You must enter only numbers.'
          :ElseIf ∨/'¯-'∊Input ⋄ ErrMsg←'Negative numbers not allowed.'
          :Else                                      ⍝ see what button pressed, call correct fns, set plot, set error message
              ⍎'file←Tri',Action,'⍎Input'            ⍝ call program using Button label: TriSSS or TriAAS etc using user Input
              GraphHtml←⊃,/{'svg'≡¯3↑⍵:'<embed width="400" height="400" src="/',⍵,'" type="image/svg+xml" />' ⋄ ''}¨file
              ErrMsg←⊃,/{'svg'≡¯3↑⍵:'' ⋄ Action,': ',enlist⍕¨⍵}¨file
          :EndIf
          Action←''                                  ⍝ reset action to nothing
      :EndSelect
    ∇

    ∇ z←RainProIn z  ⍝ workaround for no result command below not accepted in dfn
      (↑'ch' 'PostScrp' 'svg')⎕CY'rainpro' ⍝ Bring in rainpro if using MiServer
    ∇
      TriPlotSSS←{α←⊃,/'side lengths: a=' ' b=' ' c=',¨⍕¨⍵
          'J'∊⍕⍵:'INVALID ∆ (imaginary side length(side with J in it:',⍕α
          max≥(+/⍵)-max←⌈/⍵:'INVALID ∆ (longest side)≥(sum other 2 sides):',⍕α
          xy←↑(0 0)(⍵[1]0)(xy4C ⍵)  ⍝ ∆coords:A(0,0) B(a,0) C(from xy4C fns)
          z←RainProIn 0                 ⍝ Use if MiServer  (ie Web page)
          z←ch.Set'Head'(Action,' Triangle Plot:abc=sides & ABC=angles')
          z←ch.Set'ffont' 'Ar,12,blue'
          z←ch.Set'Footer'α
          cap←'Side a: horizontal. Angles at:'                         ⍝ label line
          cap,←(⊃,/¯1⌽' ',¨'BCA',¨'(',¨(⍕¨2 round¨↓xy),¨')'),';X axis' ⍝ label Angles
          z←ch.Set¨('xcap'cap)('ycap' 'Y axis')
          z←ch.Set'style' 'XYPLOT,GRID,lines,markers,filled' ⍝ set up plot
          z←ch.SetMarkers'Bullet'         ⍝ bullet symbol from ch.∆markers
          z←ch.Set'Xrange'(ran←(⌊/,xy)(⌈/,xy)) ⍝ min & max for plots
          z←ch.Set'Yrange'ran           ⍝ same x & y so it looks good
          z←ch.Plot xy                  ⍝ plot triangle
          ⍝ PG←#.ch.Close               ⍝ save plot (Use if no Miserver)
          ⍝ #.View PG                   ⍝ show plot (Use if no Miserver)
          ⍝                                         (Use below 4 lines if MiServer)
          (tn file)←'svg'#.Files.CreateTemp req.Server.TempFolder
          z←file svg.PS ch.Close
          file←(⍴req.Server.Root)↓file  ⍝ Make relative file name
          file
      }
```

NOTES FOR ALL TRIANGLE EQUATIONS AND APL SOLUTIONS *****



SYMBOLS USED: (http://www.mathsisfun.com/algebra/trig-solving-practice.html)
A B C are angles in degrees and a b c are side lengths opposite those angles. Ar Br Cr are angles in radians which APL often uses. **○1** is pi(π) in APL. A radian is a way of expressing an angle in terms of a circle's radius.
1 Radian=180°÷π. or about 57.2958 degrees(180÷3.141592654) & 57.29581× π =180 degrees

PRELIMINARY FORMULAS as programs:

```
 DegToRad←{⍵×○1÷180}    ⍝ ○1 is pi in APL, so DegToRad 57.2958 would result in 1
 RadToDeg←{⍵÷○1÷180}    ⍝ and RadToDeg 1 would result in 57.2958
 sin←{1○DegToRad ⍵}     ⍝ 1○ is sine fns in APL so sin 30 finds sine of 30deg
 cos←{2○DegToRad ⍵}     ⍝ 2○ is cosine  so cos 45 finds cosine of 45deg
 arcsin←{RadToDeg ¯1○⍵} ⍝ convert sine   of angle in radians to angle in degrees
 arccos←{RadToDeg ¯2○⍵} ⍝ convert cosine of angle in radians to angle in degrees
```

1)Triangle Angles Add to 180 degrees:
 If we have 2 angles we can get the 3rd because their sum=180.
 A+B+C=180 degrees so in APL **C←180-A+B** or **B←180-A+C** or **C←180-A+B**

2)Law of Sines:
 So if we have couple of angles and a side or a couple of sides and an angle we can find other side. (ie if we have A and a and B we can determine b)
 (a÷sin A)=(b÷sin B)=(c÷sin C) or reciprocals:**((sin A)÷a)=((sin B)÷b)=((sin C)÷c)**

3)Law of Cosines:
 (c*2)=(a*2)+(b*2) for right triangle
 (c*2)=(a*2)+(b*2)-2×a×b×cos C for any triangle C in degrees

4)Area of a triangle:

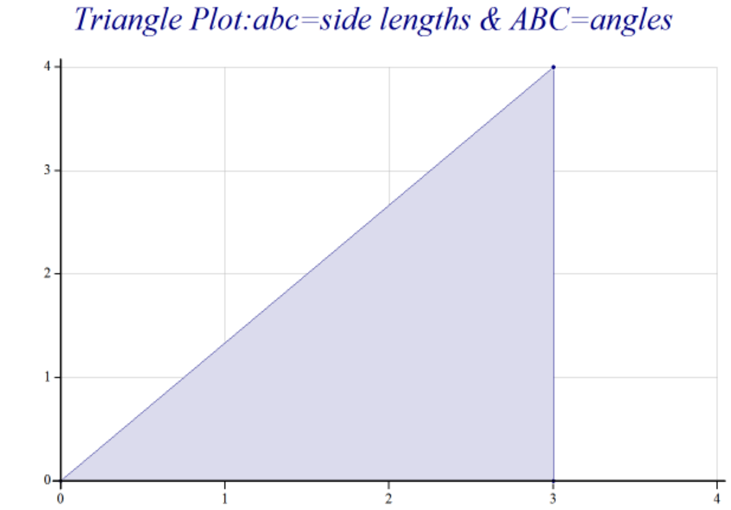
```
 area←×/0.5 a b(sin C) ⍝ or for a right triangle C=90° & sin 90 =1 so area =.5×a×b×1
```

So with these 4 basic formulas we can solve all triangle problems

HERE ARE 7 FUNCTIONS THAT SOLVE ALL POSSIBLE TRIANGLE PROBLEMS: A=angle S=side
 TriAA TriAAA TriAAS TriASA TriSAS TriSSA TriSSS

EXAMPLE USAGE: capitals A B C are angles. Small letters a b c are side lengths
If a triangle had 3 sides: 3,4,5 do this:

```
     TriSSS 3 4 5  ⍝ Type in 3 sides(3,4,5) & get all angles & area info back.
```

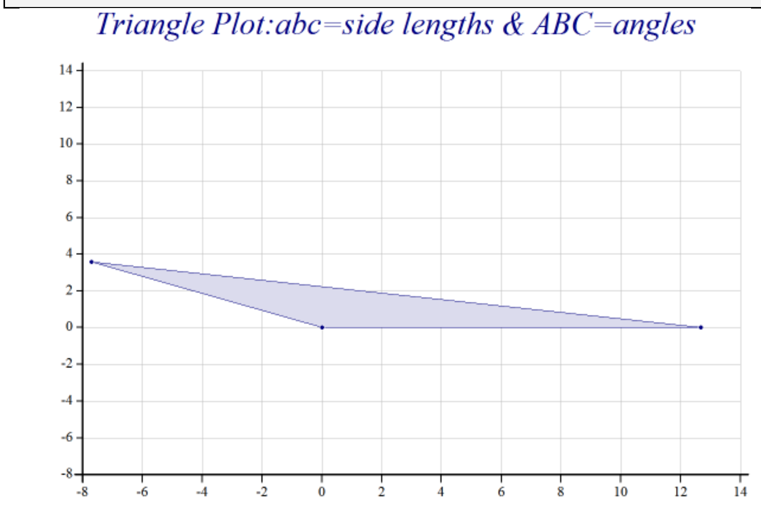


Triangle Plot:abc=side lengths & ABC=angles

a b c= 3 4 5 A B C= 36.8699 53.1301 90 area= 6

If a triangle had 2 angles and a side 10 degrees 15 degrees and side 8.5 do this:

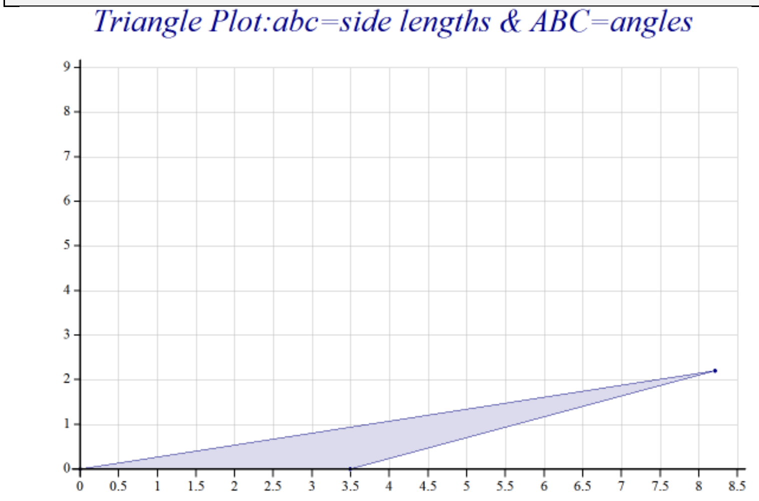
```
        TriAAS 10 15 8.5
```



```
a b c= 12.6691 20.687 8.5 A B C= 15 155 10 area= 22.7553
```

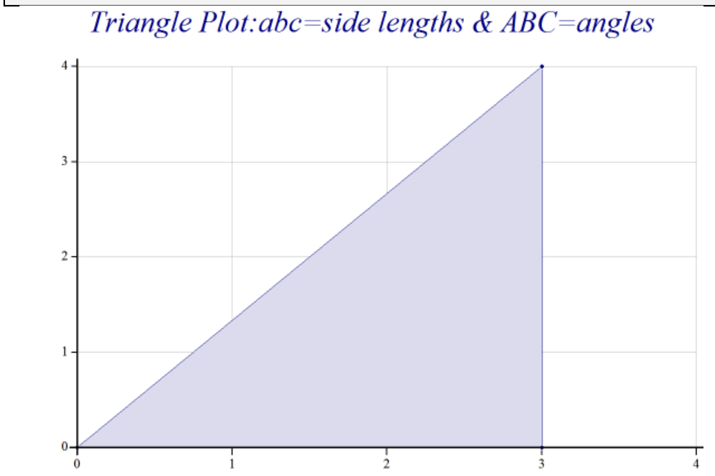
If a triangle had an angle 10, then a side 8.5 and then an angle 15 do this:

```
        TriASA 10 8.5 15
```



```
a b c= 3.4925 5.2056 8.5 A B C= 10 15 155 area= 3.8417
```

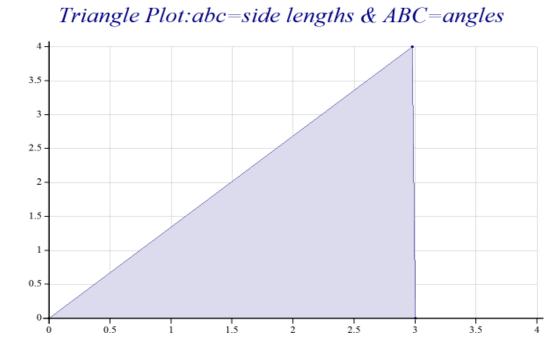
```
        TriSAS 3 90 4                         A this is right triangle so a*2 + b*2 = c*2
```



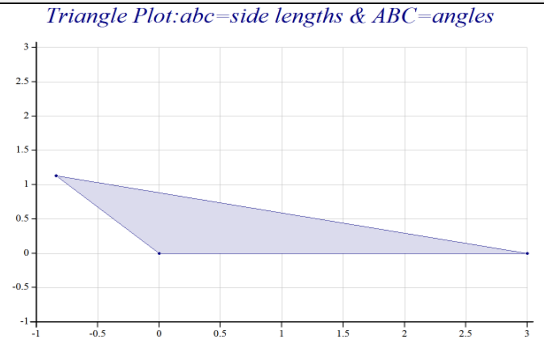
```
a b c= 3 4 5 A B C= 36.8699 53.1301 90 area= 6
```

A area is (a×b)÷2

```
      TriSSA 3 4 37 ⍝ there are two possible solutions for this problem
```



a b c= 3 4 4.9848 A B C= 37 53.3618 89.6382 area= 5.9999 [Alt 1]



a b c= 3 4 1.4043 A B C= 37 126.6382 16.3618 area= 1.6902 [Alt 2]

There are a number of triangles which are impossible, angles cannot sum to more than 180 degrees & one side cannot be longer than the sum of the other two sides.

```
      TriAAS 100 95 8.5
INVALID ∆ 2 input angles sum≥180: 100 95
```

```
      TriSSS 3 4 8  ⍝ impossible triangle c>a+b
INVALID ∆ (longest side)≥(sum other 2 sides):a b c= 3 4 8
```

Finally some combinations of angles and sides are not possible as indicated in the example below where **TriSSA** finds imaginary numbers noted in APL with **J**. **TriSAS 3 90 4** is the 3 4 5 right triangle, but **TriSSA 3 4 90** is impossible in two different ways as is show below.

```
      TriSSA 3 4 90 ⍝ many other combs/orders of angles & sides are also invalid.
INVALID ∆ (imaginary side length(side with J in it:a b c= 3 4 0J2.6458 A B C= 90
90J¯45.5711 0J45.5711 area= 0J5.2915 [Alt  1 ]
INVALID ∆ (imaginary side length(side with J in it:a b c= 3 4 0J¯2.6458 A B C= 90
90J45.5711 0J¯45.5711 area= 0J¯5.2915 [Alt  2 ]
```

HERE ARE THE ACTUAL FUNCTIONS:

```
TriAA←{⍝ Triangle info given AngleAngle
     C←180-+/A B←⍵ ⍝ input: 2 angles A B
     in←(C<0)/'INVALID ∆ 2 input angles sum≥180:'
     in,'A B C=',A,B,C,'a b c area=Need at least 1 side to do more'}

TriAAA←{⍝ Triangle info given AngleAngleAngle
     in←(180≠+/⍵)/'INVALID ∆ 3 input angles not equal to 180:'
     in,'A B C=',⍵,'a b c area=Need at least 1 side to do more'}

TriAAS←{⍝ Triangle info given Angle Angle Side
     C A c←⍵ ⍝ A C=angles c=side opposite angle C
     B←180-+/A C            ⍝ missing angle B=180-(A+C)
     B<0:'INVALID ∆ 2 input angles sum≥180:',C,A
     ⍝ law of sines is (a÷sin A)=(b÷sin B)=(c÷sine C)
     a←(c×sin A)÷sin C      ⍝ solve law of sines for a
     b←(c×sin B)÷sin C      ⍝ solve law of sines for b
     area←×/0.5 a b(sin C) ⍝ .5×base×ht [ht=b×sin C]
     ('a b c=',(4 round a b c),'A B C=',(4 round A B C),'area=',4 round
area)TriPlotSSS a b c}

TriASA←{⍝ Triangle info given AngleSideAngle
     A c B←⍵ ⍝ A C=angles c=side opposite angle C
     C←180-+/A B            ⍝ missing angle C=180-(A+B)
     C<0:'INVALID ∆ 2 input angles sum≥180:',A,B
     ⍝ recall law of sines: (a÷sin A)=(b÷sin B)=(c÷sine C)
     a←(c×sin A)÷sin C     ⍝ solve sine law for a using C
     b←(c×sin B)÷sin C     ⍝ solve sine law for b using C
     area←×/0.5 a b(sin C) ⍝ .5×base×ht [ht=b×sin C]
     ('a b c=',(4 round a b c),'A B C=',(4 round A B C),'area=',4 round
area)TriPlotSSS a b c}

TriSAS←{⍝ Triangle info given SideAngleSide
     a C b←⍵ ⍝ a=side1 C=angle between b=side2
     c←0.5*⍨(+/a b*2)-(×/2 a b)×cos C ⍝ c=sqrt(a2+b2 - 2ab×Cos C
     ⍝ Note: law of sines (sin A/a) = (sin B/b) = (sin C/c)
     SinAr←(a×sin C)÷c     ⍝ solve sine law for sine A(in radians)
     A←arcsin SinAr        ⍝ convert sine A in radian to angle in deg
     B←180-+/A C           ⍝ missing angle B=180-(A+C)
     area←×/0.5 a b(sin C) ⍝ .5×base×ht [ht=b×sin C]
     ('a b c=',(4 round a b c),'A B C=',(4 round A B C),'area=',4 round
area)TriPlotSSS a b c}

TriSSA←{⍺←0 ⍝ Triangle info given SideSideAngle. There are 2 possible triangles
     a b A←⍵ ⍝ a=side opposite angle A  b=side ⍝ Note: this program runs twice
     Ar←DegToRad A         ⍝ convert A to radians
     ⍝ recall law of sines is : (a÷sin Ar)=(b÷sin Br)=(c÷sin Cr)
     ⍝ solve law of sines for sin b: sin b=(b×sin a)÷a
     SinBr←(b×sin A)÷a     ⍝ solve sine law for sin of B(in radians)
     B←arcsin SinBr        ⍝ convert sine of Br in radians to B in degrees
     B←(⍺+1)⊃B,180-B       ⍝ pick B(⍺=0) or 180-B(⍺=1) for 2 possible b angles
     C←180-+/A B           ⍝ ⍝ missing angle C=180-(A+B)
     c←(b×sin C)÷sin B     ⍝ solve law of sin's for c=(b×sin C)÷sin B
     area←×/0.5 a b(sin C) ⍝ .5×base×ht [ht=b×sin C]
     ⎕←('a b c=',(4 round a b c),'A B C=',(4 round A B C),'area=',(4 round
area),'[Alt ',(⍺+1),']')TriPlotSSS a b c
     ⍺=0:1 ∇ ⍵}  ⍝ call TriSSA (∇) again with same inputs(⍵) but ⍺=1 picks 180-B}

TriSSS←{⍝ Triangle solution given 3 sides
     a b c←⍵                                ⍝ input: 3 sides
     ⍝ note:arccosine=¯2○ It converts cosine to angle in radians
     ⍝ recall cosine fns is: (a*2)=(b*2)+(c*2)-2×b×c×cosine Ar
     A←arccos((+/(b c)*2)-a*2)÷×/2 b c ⍝ Cosine function solved for A
     B←arccos((+/(c a)*2)-b*2)÷×/2 c a ⍝ Cosine function solved for B
     C←180-+/A B                       ⍝ missing angle C=180-(A+B)
     area←×/0.5 b c(sin A)             ⍝ .5×base×ht [ht=c×sin A]
     ('a b c=',(4 round a b c),'A B C=',(4 round A B C),'area=',4 round
area)TriPlotSSS a b c}
```

Bingo ****

Imagine 5x5 Bingo game where Bingo numbers are determined by simple math(2 numbers added, subtracted, multiplied or divided). For example the caller might say “2 times 4” and if you had an 8 on your board you would put an X over the 8. What would be the best numbers(1-50 no duplicates) for you to place on your board? Well add and subtract are unbiased but for multiply and divide some numbers have more *factors* and thus will occur more often. Lets find best numbers to put on your board so you can win the Bingo game.

```
      factors←{(r=⌊r←⍵÷n)/n←⍳⌊⍵÷2} ⍝ fns to find all factors for a number.
      factors 30    ⍝ call fns factors passing 30 into the program(⍵)
```

```
1 2 3 5 6 10 15      ⍝ these are the factors of 30
```

Let me explain the above **factors** program from right to left. **ω** which is 30 is ÷2(since no factor can be greater than ½ the number). **⌊** rounds the number down if it is a decimal and **ι** makes the numbers from 1-15 and stores them in **n**. **r** is **ω**(30)÷each **n**(numbers 1-15). **⌊**rounds the results(**r**) down and = compares each **r** to it's rounded **r**. If **r=⌊r** the division must have come out even and thus **n** must be a factor. The expression inside the () will be 15 1's and 0's showing which values of **n** are factors of 30. The syntax **(r=⌊r)/n** selects only **n**'s which have 1's. Here 30÷1 2 3 … 15 has even results for 1 2 3 5 6 10 15 which are the factors for 30.

Now lets find all factors for each(¨) number 1-50(ι**50**). Then catenate(,) factors with each(¨) of the numbers and make a table(**⍉↑**) for viewing.

```
       ⍉↑(ι50),¨factors¨ι50    ⍝ row 1 is the #, other rows are the factors
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
0 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
0 0 0 2 0 2 0 2 3  2  0  2  0  2  3  2  0  2  0  2  3  2  0  2  5  2  3  2  0  2  0  2  3  2  5  2  0  2  3  2  0  2  0  2  3  2  0  2  7  2
0 0 0 0 0 3 0 4 0  5  0  3  0  7  5  4  0  3  0  4  7 11  0  3  0 13  9  4  0  3  0  4 11 17  7  3  0 19 13  4  0  3  0  4  5 23  0  3  0  5
0 0 0 0 0 0 0 0 0  0  0  4  0  0  0  8  0  6  0  5  0  0  0  4  0  0  0  7  0  5  0  8  0  0  0  4  0  0  0  5  0  6  0 11  9  0  0  4  0 10
0 0 0 0 0 0 0 0 0  0  0  6  0  0  0  0  0  9  0 10  0  0  0  6  0  0  0 14  0  6  0 16  0  0  0  6  0  0  0  8  0  7  0 22 15  0  0  6  0 25
0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0 10  0  0  0  0  0  9  0  0  0 10  0 14  0  0  0  0  0  8  0  0
0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0  0  0 15  0  0  0  0  0 12  0  0  0 20  0 21  0  0  0  0  0 12  0  0
0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 18  0  0  0  0  0  0  0  0  0  0  0 16  0  0
0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 24  0  0
```

Looking at the table we can see that 48 has the most factors and odd numbers generally are much poorer than even numbers. Now lets put these results in order by the number of factors(ρ). First lets get counts:

```
+m←⍉↑(ι50),¨ρ¨factors¨ι50      ⍝ row 1 is the #, row 2 is the # of factors
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
0 1 1 2 1 3 1 3 2  3  1  5  1  3  3  4  1  5  1  5  3  3  1  7  2  3  3  5  1  7  1  5  3  3  3  8  1  3  3  7  1  7  1  5  5  3  1  9  2  5
```

```
       m[;⍒m[2;]] ⍝ Descending Sort(⍒) using row 2 of m to sort m
48 36 24 30 40 42 12 18 20 28 32 44 45 50 16 6 8 10 14 15 21 22 26 27 33 34 35 38 39 46 4 9 25 49 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 1
 9  8  7  7  7  7  5  5  5  5  5  5  5  5  4 3 3  3  3  3  3  3  3  3  3  3  3  3  3  3 2 2  2  2 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1 0
```

In the above **m** is a matrix with 2 rows and 50 columns. **m[rows;columns]**. So **⍒m[2;]** takes row 2 values of matrix & determines their reverse sort order.

```
48 36 24 30 40 42 12 18 20 28 32 44 45 50 16 6 8 10 14 15 21 22 26 27 33 34 35 38 39 46 4 9 25 49 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 1
```

so highest value is col 48 which is in this case 48. Worst value is in column 1 which is 1. So now you can pick best values for your Bingo game easily. I would suggest putting best values all in same row or column. So your first row or column might be 48 36 24 30 40 and next row/column might then be 42 12 18 20 28 etc. These would have highest odds of winning.
Now verify by testing if this is correct. Here's fns that makes 3 different Bingo cards(numbers with fewest, random or most factors) and then evaluates them with random product numbers and sees which card wins(5 in row).

```
res←Bingo ss;FactorCalc;facts;boardL;boardM;boardR;calls;prods;RCMAX;n;z;bL;bR;bM
⍝ Evaluate 5×5 Bingo games with boards #'s 4-50 constructed in 3 ways:
⍝    1)Least factors, 2)Most factors 3)Random #'s
⍝ show steps ss=( 0:no 1:partial 2: play mode-full details and pause at each step)
⍝      use: +/5=⍎¨1000ρ⊂'Bingo 0'  to test program 1000 times and show winner boards
⍝ Bingo calls are determined by multiplying 2 random numbers 2-25
 FactorCalc←{(r=⌊r←ω÷nums)/nums←ι⌊ω÷2} ⍝ fns to determine factors
 facts←⊃,/ρ¨FactorCalc¨3+ι47 ⍝ get factors for #'s 1-50
 boardL←5 5ρ3+⍋facts          ⍝ 1)board with #'s with least factors (sort up)
 boardM←5 5⍴3+⍒facts           ⍝ 2)board with #'s with most  factors (sort down)
 boardR←5 5⍴3+25?47          ⍝ 3)board with #'s random(?) 4-50 no duplicates
```

```
⍝ n unique(∪) product(×/¨) #'s <50 from 5000 random(?) pairs of #'s (2-25)
```

```
 prods←×/¨calls←∪(50>×/¨calls)/(calls←1+?5000⍴⊂24 24) ⍝ gen random product Bingo calls
 calls←calls[prods⍳∪prods] ⋄ prods←∪prods ⍝ keep only calls with unique(∪) products
 RCMAX←{(⌈/+/⍵)⌈(⌈/+⌿⍵)}   ⍝ fns:Row Col MAX: ⍵ is input i.e. 5×5 board
                           ⍝ fns gets largest(⌈/) rowsum(+/) or colsum(+⌿)
 :For n :In ⍳⍴calls        ⍝ loop :For each call:count each boards matches
     res←RCMAX¨(bL bR bM←boardL boardR boardM∊¨⊂n↑prods) ⍝ score each board for trial
     :If ss>0 ⋄ '  Least=' '   Random=' '    Most=' 'Hits for Trial=' '#=',¨res,n,calls[n]
     :EndIf
     :If ss>1
         ⍞←'Enter to see trial results or b:see full boards or q:quit ' ⋄ z←¯1↑⍞ ⍝ ask&wait
         '|',¨bL bR bM×boardL boardR boardM   ⍝ show scored boards each step if ss>1
         :If z≡,'b'
             '   Least Factors     Random# Factors   Most Factors' ⋄ '|',¨boardL boardR boardM
         :ElseIf z≡,'q' ⋄ →0 ⍝ exit(go to zero) if response is "q"
         :EndIf
     :EndIf
     →0×⍳5∊res ⍝ exit(go to zero) if any board wins:row/col sum matches(∊) a 5 ⍝ to Play through
 :EndFor
```

Now lets play Bingo by trying the Bingo fns a couple times.

```
     Bingo 1
Least= 0 Random= 1 Most= 1 Hits for Trial= 1 #=2 8
Least= 1 Random= 1 Most= 1 Hits for Trial= 2 #=2 3
Least= 1 Random= 1 Most= 1 Hits for Trial= 3 #=2 17
Least= 1 Random= 1 Most= 2 Hits for Trial= 4 #=11 4
Least= 1 Random= 2 Most= 2 Hits for Trial= 5 #=4 3
Least= 1 Random= 2 Most= 2 Hits for Trial= 6 #=3 16
Least= 1 Random= 2 Most= 2 Hits for Trial= 7 #=13 2
Least= 1 Random= 2 Most= 3 Hits for Trial= 8 #=9 4
Least= 1 Random= 2 Most= 3 Hits for Trial= 9 #=6 4
Least= 1 Random= 2 Most= 3 Hits for Trial= 10 #=3 7
Least= 1 Random= 2 Most= 4 Hits for Trial= 11 #=4 8
Least= 1 Random= 2 Most= 5 Hits for Trial= 12 #=2 21
1 2 5
```

```
Bingo 1
Least= 0 Random= 0 Most= 1 its for Trial= 1 #= 3 7
Least= 0 Random= 0 Most= 1 its for Trial= 2 #= 2 20
Least= 0 Random= 0 Most= 2 its for Trial= 3 #= 18 2
Least= 0 Random= 1 Most= 2 its for Trial= 4 #= 4 11
Least= 1 Random= 1 Most= 2 its for Trial= 5 #= 2 2
Least= 1 Random= 1 Most= 3 its for Trial= 6 #= 8 3
Least= 1 Random= 1 Most= 3 its for Trial= 7 #= 2 16
Least= 1 Random= 1 Most= 4 its for Trial= 8 #= 3 16
Least= 1 Random= 1 Most= 4 its for Trial= 9 #= 2 8
Least= 1 Random= 1 Most= 4 its for Trial= 10 #= 19 2
Least= 1 Random= 1 Most= 5 Hits for Trial= 11 #= 5 6
1 1 5
```

As you can see the board using numbers with the **Most** factors won both times. I tested this 1000 calls: **+/5=⍎¨1000⍴⊂'Bingo 0'** and got:11 61 952. So **Most** wins (or ties) 95.2%(952÷1000) of the time.

In the game originally described not all calls are made from multiplication. Some were also made from addition, subtraction and division. Addition would have bias towards larger numbers while subtraction would have bias towards smaller numbers but overall advantage for boards with more factors would be smaller. What is the bias for division?

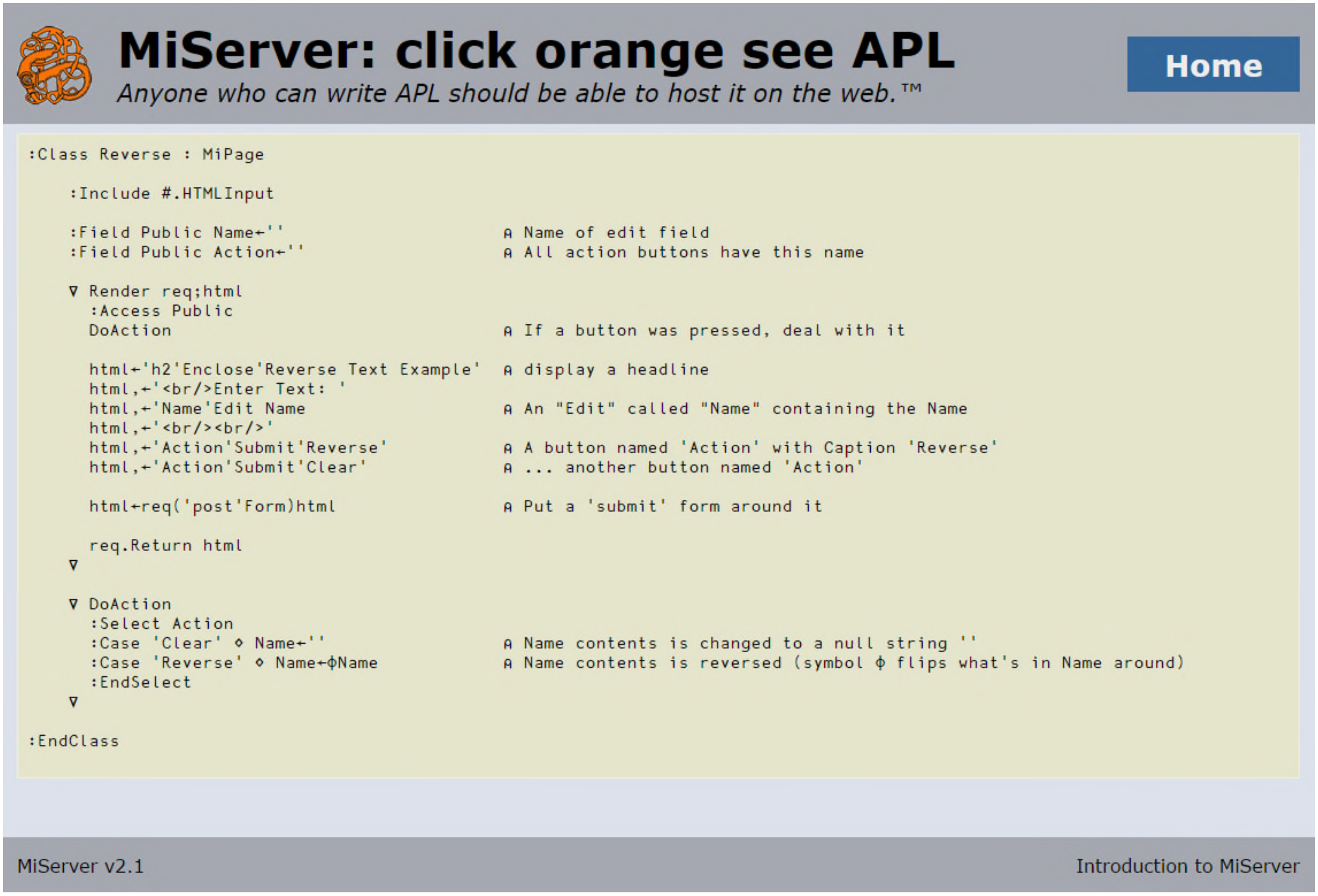
If you call the program like this:

```
Bingo 2
```

It will play in an interactive mode where you can watch each of 3 boards be scored at each step.

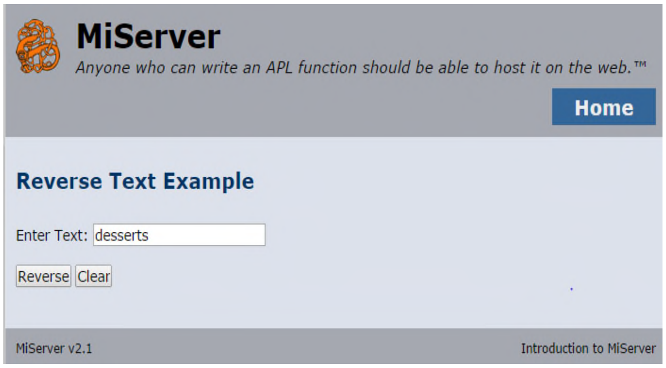
Writing Web page using APL Using Mildserver ***

An APL **Class** is created called **Reverse**. Automatic Code(**MiPage** & **HTMLInput**) is included which does most of the work creating webpage & converting APL to HTML in the **Render** fns. **DoAction** fns checks which **Action** button was pressed **Clear** or **Reverse** & does what Submit **Caption** says: If '**Reverse**' letters in **Name** reversed **Name←⌽Name**. If '**Clear**' **Name** set to null **Name←''**.


MiServer: click orange see APL
Anyone who can write APL should be able to host it on the web.™
Home

```
:Class Reverse : MiPage

    :Include #.HTMLInput

    :Field Public Name←''                        ⍝ Name of edit field
    :Field Public Action←''                      ⍝ All action buttons have this name

    ∇ Render req;html
      :Access Public
      DoAction                                   ⍝ If a button was pressed, deal with it

      html←'h2'Enclose'Reverse Text Example'     ⍝ display a headline
      html,←'<br/>Enter Text: '
      html,←'Name'Edit Name                      ⍝ An "Edit" called "Name" containing the Name
      html,←'<br/><br/>'
      html,←'Action'Submit'Reverse'              ⍝ A button named 'Action' with Caption 'Reverse'
      html,←'Action'Submit'Clear'                ⍝ ... another button named 'Action'

      html←req('post'Form)html                   ⍝ Put a 'submit' form around it

      req.Return html
    ∇

    ∇ DoAction
      :Select Action
      :Case 'Clear' ⋄ Name←''                    ⍝ Name contents is changed to a null string ''
      :Case 'Reverse' ⋄ Name←⌽Name               ⍝ Name contents is reversed (symbol ⌽ flips what's in Name around)
      :EndSelect
    ∇

:EndClass
```

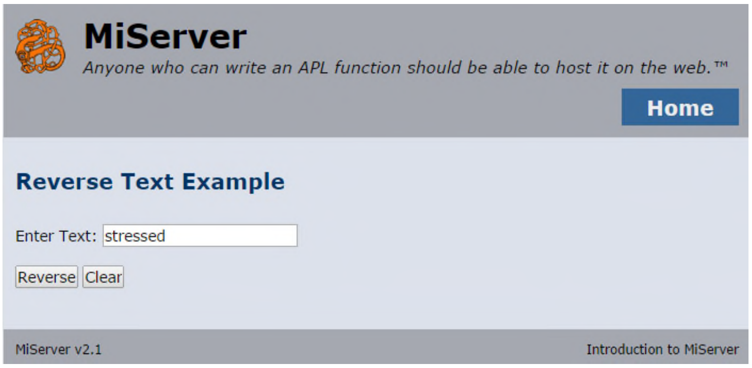
Below is **Web Page before & after** you press **Reverse** button. Notice reversed text. If you pressed **Clear** button Text would be erased & pressing **Home** changes webpage to the parent webpage. To see goto **jerrymbrennan.com** click **APL Apps on MiServer** at bottom of page, then **ALL** then **Simple MiPage with form.** Click the orange snake to see the above code and again to see below code.


MiServer
Anyone who can write an APL function should be able to host it on the web.™
Home

Reverse Text Example

Enter Text: desserts

Reverse
Clear
MiServer v2.1
Introduction to MiServer


MiServer
Anyone who can write an APL function should be able to host it on the web.™
Home

Reverse Text Example

Enter Text: stressed

Reverse
Clear
MiServer v2.1
Introduction to MiServer

APL References & Info About My Website And Access To It

For educational use you can get a free version of this APL at: http://dyalog.com/ This includes everything. There are thousands of pages of online manuals and tutorials describing everything available.

Eight Intro Dyalog APL education videos: Do APL101-APL108 first. https://www.youtube.com/playlist?list=PL1955671BD6E21548

Online Dyalog APL tutorial with a sandbox where you can try out lines of APL code such as from this tutorial except for the plotting things. www.tryapl.org

Complete APL tutorial(not Dyalog specific) with a sandbox at: http://aplwiki.com/LearnApl/LearningApl

Repository of articles, videos and tutorials about APL: http://aplwiki.com

Video shows **Game of Life** in **APL**. Video demos the amazing power & conciseness of **APL**. http://www.youtube.com/watch?fmt=18&gl=GB&hl=en-GB&v=a9xAKttWgP4

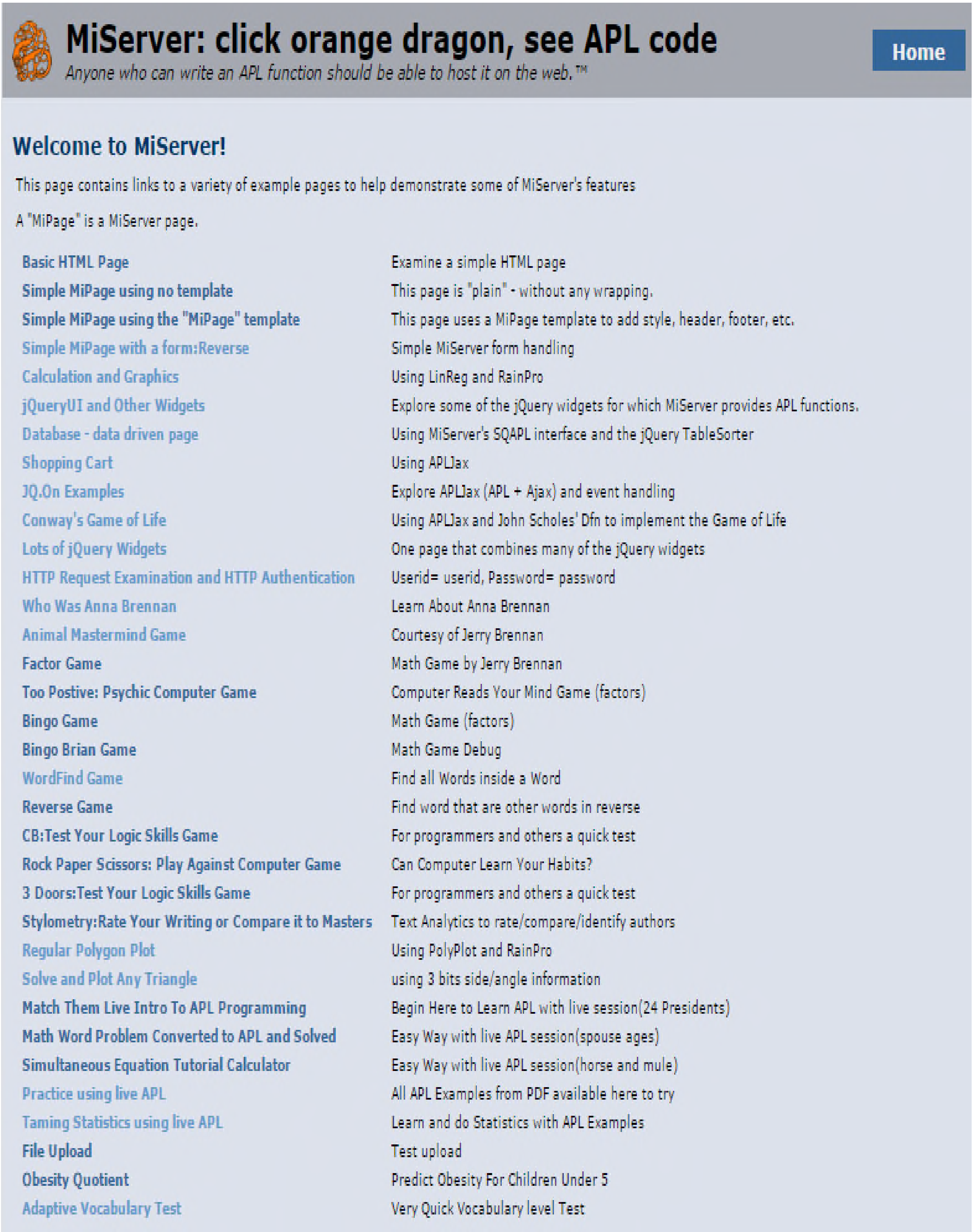
More educational videos at: http://www.youtube.com/user/APLtrainer

Extensive(800+ pages) Dyalog APL tutorial book you can download for free http://dyalog.com/mastering-dyalog-apl.htm or http://dyalog.com/uploads/documents/MasteringDyalogAPL.pdf

http://en.wikipedia.org/wiki/APL_(programming_language) A Programming Language (APL). History and advantages of APL described.

Some information about **Kenneth E. Iverson** the inventor of APL. He was a Harvard Mathematics Professor, worked for IBM and won a Turing Award for creating APL. He first developed APL as a concise notation for mathematics. Later he developed it as a comprehensive computer language. http://en.wikipedia.org/wiki/Kenneth_E._Iverson

My APL Educational Web page. Goto: http://JMB.APLCloud.com or my web page http://jerrymbrennan.com/ & click on **APL Lessons using MiServer** at page bottom to see menu below of many interactive example APL web pages of games, lessons and math and language utilities. Click orange dragon upper left on every page to see actual APL code for that page & Home button takes you back to main menu. Click **Practice using live APL** below to try all the examples in this handout yourself or do anything else. Play numerous games, watch videos, do many interactive tutorials and learn about your logical thinking errors and then see the actual code that created everything. (**SEE NEXT PAGE FOR MAIN MENU Note: there are many submenus also**)



MiServer: click orange dragon, see APL code

Anyone who can write an APL function should be able to host it on the web.™

Welcome to MiServer!

This page contains links to a variety of example pages to help demonstrate some of MiServer's features

A "MiPage" is a MiServer page.

Basic HTML Page	Examine a simple HTML page
Simple MiPage using no template	This page is "plain" - without any wrapping.
Simple MiPage using the "MiPage" template	This page uses a MiPage template to add style, header, footer, etc.
Simple MiPage with a form:Reverse	Simple MiServer form handling
Calculation and Graphics	Using LinReg and RainPro
jQueryUI and Other Widgets	Explore some of the jQuery widgets for which MiServer provides APL functions.
Database - data driven page	Using MiServer's SQAPL interface and the jQuery TableSorter
Shopping Cart	Using APLJax
JQ.On Examples	Explore APLJax (APL + Ajax) and event handling
Conway's Game of Life	Using APLJax and John Scholes' Dfn to implement the Game of Life
Lots of jQuery Widgets	One page that combines many of the jQuery widgets
HTTP Request Examination and HTTP Authentication	Userid= userid, Password= password
Who Was Anna Brennan	Learn About Anna Brennan
Animal Mastermind Game	Courtesy of Jerry Brennan
Factor Game	Math Game by Jerry Brennan
Too Postive: Psychic Computer Game	Computer Reads Your Mind Game (factors)
Bingo Game	Math Game (factors)
Bingo Brian Game	Math Game Debug
WordFind Game	Find all Words inside a Word
Reverse Game	Find word that are other words in reverse
CB:Test Your Logic Skills Game	For programmers and others a quick test
Rock Paper Scissors: Play Against Computer Game	Can Computer Learn Your Habits?
3 Doors:Test Your Logic Skills Game	For programmers and others a quick test
Stylometry:Rate Your Writing or Compare it to Masters	Text Analytics to rate/compare/identify authors
Regular Polygon Plot	Using PolyPlot and RainPro
Solve and Plot Any Triangle	using 3 bits side/angle information
Match Them Live Intro To APL Programming	Begin Here to Learn APL with live session(24 Presidents)
Math Word Problem Converted to APL and Solved	Easy Way with live APL session(spouse ages)
Simultaneous Equation Tutorial Calculator	Easy Way with live APL session(horse and mule)
Practice using live APL	All APL Examples from PDF available here to try
Taming Statistics using live APL	Learn and do Statistics with APL Examples
File Upload	Test upload
Obesity Quotient	Predict Obesity For Children Under 5
Adaptive Vocabulary Test	Very Quick Vocabulary level Test