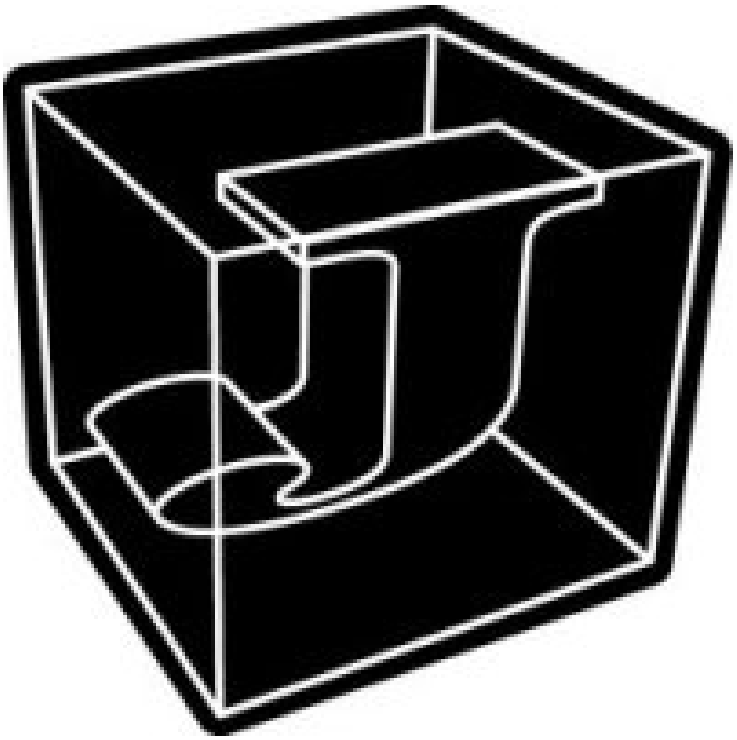
Exploring Math



Kenneth E. Iverson

Copyright © 1996-2002 Jsoftware Inc. All rights reserved.

Table Of Contents

Chapter 1

Exploration 1
A. Introduction 1
B. Ramble Or Research 6

What Is Math 9
A. Relations 9
B. Proofs 13
C. Summary 14

Function Tables 17

Grammar And Spelling 23
A. Introduction 23
B. The Use Of Grammar 24
C. Punctuation And Other Rules 25
D. Spelling 27

Reports 29
A. Introduction 29
B. Transposition 31

Terminology 33

Decimal and Other Number Systems 37
A. Introduction 37
B. Addition 42
C. Multiplication 44
D. Subtraction 45

Recursion 47

Proofs 53
A. Introduction 53
B. Inductive Proof 56

Tools 61
A. Introduction 61
B. Editing 62
C. Script Windows 62

Coordinates and Visualization 63
A. Introduction 63
C. Plotting Multiple Figures 67
D. Plotting Functions 68

Linear Functions 73
A. Distributivity 73
B. Linearity 74
C. Linear Vector Functions 75
D. Inner Product 76
E. Why The Name “Linear”? 77

Representations of Functions 81
A. Introduction 81

Polynomials 85
A. Coefficients Representation 85
B. Roots Representation 86
C. Versatility 87
D. Parity 89
E. Linearity 90
F. Polynomial Approximations 92

Arithmetic 95
A. Introduction 95
B. Insidious Inverses 95
C. Rational Numbers 96
D. Irrational Numbers 97
E. Complex Numbers 97

Complex Numbers 99
A. Introduction 99
B. Addition 100
C. Multiplication 101
D. Powers and Roots 103
E. Division 104

Calculus 107
A. Secant Slope 107
B. Derivative 108
C. Polynomials 109
D. Differential Equations 110
E. The Exponential Family 112

Inverses and Equations 115
A. Inverse Functions 115
B. Monotonic Functions 116
C. Under 117
D. Equations 118

Readings 121
A. Introduction 121
B. Phrases 121
C. Sample Topics 121
D. Vocabulary and Definitions 122

References **123**

Index **125**

Chapter 1

Exploration

Something lost behind the ranges
Lost and waiting for you. Go!
Kipling

A. Introduction

Exploring a city or wild park on foot is more fun, and often more instructive, than studying it in books, lectures, or pictures. A map or other guide may be helpful, but it is important to be able to experiment, choosing your own path, approaching points of interest from various directions. This can give you a sense of the lay of the land that is more useful, and more lasting, than any fixed tour of "important points" laid out by someone else.

Matters other than landscapes may also be explored, effectively and enjoyably. For example, to learn about clockwork, begin not with diagrams and discussions of balance wheels, springs, and escapements, but rather with an actual old-style, wind-up alarm clock. Explore it by first finding what can be done with it. Can you: reset the time? make it run faster? stop it? or reset the hour hand independently of the minute hand?

Having learned what it can do, explore the matter of how it does it, by removing its cover, studying the works, and finally taking it apart and re-assembling it. You may, of course, not be skillful enough to get it working again.

Exploration can also be applied to other devices that may be more interesting or more easily available to you: toasters, typewriters, electrical toggle switches, or door locks. But do not forget your own safety—danger lurks in electrical devices as well as in wilderness parks. Finally, in choosing a device for exploration, favour the older models: modern typewriters and digital clocks may be totally inscrutable. At least one author (Ivan Illich) has claimed to see a sinister motive in this, claiming that modern design is deliberately inscrutable in order to keep ordinary people like us in ignorance.

But can exploration be applied to abstract, non-physical notions such as math? Yes it can. With an ordinary hand-calculator you can explore the relation between multiplication and addition by using it to multiply two by three, then to add two plus two plus two, and then comparing the results. If the calculator has a button for *power*, you can even explore that less-familiar notion by doing two to the power three, and comparing the result with two times two times two.

But the abilities of a calculator are limited, and for a general exploration of math we will use a computer equipped with suitable software called J. It is available from Website *http://www.jsoftware.com* . We will assume that you have J at hand on a computer, and will simply show examples of exploring math with it:

```
   3+2
5
   3*2
6
   3-2
1
```

These examples are in a uniformly-spaced font (Courier) that differs from the Roman font used elsewhere. We will use this difference to append comments to some of the examples. In typing the examples on your computer, enter only the part in Courier (followed by pressing the Enter key), but do not enter anything that appears in Roman. Thus:

```
   3+2                Addition
5

   three=:3           Assign the name three to 3
   three+2            Use the assigned name in a sentence
5

   b=:2
   b*b
4
```

In experiments on a sequence of numbers, it will be easier to make the entries and to compare the results if we treat them as a list. This may be illustrated as follows:

```
   2*0
0

   2*1
2
   2*2
4

   2*0,1,2,3,4,5
0 2 4 6 8 10

   a=:0,1,2,3,4,5
   2*a
0 2 4 6 8 10

   a+a
0 2 4 6 8 10
```

Comparisons can be shown more clearly by using the *equals* function as follows:

```
   (2*a)=(a+a)
```

```
1 1 1 1 1 1
```

```
   a^2
0 1 4 9 16 25
```
The list a to the power 2 (that is, the square)

```
   a*a
0 1 4 9 16 25
```

```
   (a^3)=(a*a*a)
1 1 1 1 1 1
```
The cube equals a product of three factors

Lists of integers (whole numbers) are so useful that a special function is provided for making them. Enter the following expressions, and comment on the results:

```
   i.6
0 1 2 3 4 5
```
The first six non-negative integers (whole numbers)

```
   a=:i.6
```
Read aloud as **a** ***is*** (the list) **i.6**

```
   b=:?.~6
   b
5 1 2 4 3 0
```
The integers in (repeatable) random order

```
   a+b
5 2 4 7 7 5
```

```
   a*b
0 1 4 12 12 0
```

```
   2*a
0 2 4 6 8 10
```
The even numbers (divisible by 2)

```
   1+2*a
1 3 5 7 9 11
```
The odd numbers

```
   a=b
0 1 1 0 0 0
```

As shown by the last result, the lists **a** and **b** are not equal, but they are *similar* in the sense that one can be obtained from the other by shuffling or *permuting* the items. It is rather easy to see that **a** and **b** are similar, but for longer lists similarity is not so easy to spot. For example, are the following lists similar?

```
   p=:2 15 9 10 4 0 13 13 18 7 10 16 0 1 10 13 0 7 1 8
   q=:7 4 7 13 0 10 1 1 2 13 13 15 0 10 9 18 10 8 0 16
```

A good general method for determining similarity is to first sort each list to ascending order, and then compare the results:

```
   sort=: /:~
   sort p
0 0 0 1 1 2 4 7 7 8 9 10 10 10 13 13 13 15 16 18

   sort q
0 0 0 1 1 2 4 7 7 8 9 10 10 10 13 13 13 15 16 18
```

```
   (sort p)=(sort q)
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

   (sort p)-:(sort q)
1
```

The last sentence above uses `-:` to *match* the two lists, giving `1` if they agree in every item, and `0` otherwise. This makes a comparison possible without reading all the items that result from an equals comparison.

Exercises are commonly used by a student or teacher to test a student's understanding, in order to decide what best to do next. We will also use them to suggest further exploration. A few tips on carrying out such explorations:

Before pressing the enter key, think through what the result *should* be; experiments will teach much more if this rule is always followed.

On the other hand, do not hesitate to try anything you choose; the result may be unintelligible or it may be an error message, but no serious harm can occur.

Use lists in experiments. Their results often show interesting patterns.

Do not hesitate to try things totally unknown. For example:

```
   %:a
0 1 1.41421 1.73205 2 2.23607
```

This result will probably convince you that you have discovered the symbol for the square root, and you might experiment further as follows:

```
      roots=:%:a
      roots*roots
   0 1 2 3 4 5
```

- However, do not spend too much time on results that may be, at the moment, beyond your powers. It may be better to defer further exploration until you have learned some further math (such as complex numbers). For example:

```
      %:-a
  0 0j1 0j1.41421 0j1.73205 0j2 0j2.23607
```

- Explore a complex sentence by experimenting with its parts. For example:

```
   i:4                          Function for symmetric lists
_4 _3 _2 _1 0 1 2 3 4

   i: 3
_3 _2 _1 0 1 2 3

   +:3
6

   >: +:3
7

   >:@+:3                       The function f@g is f atop (applied to the result
   of) g
7
```

```
   i.@>:@+:3
0 1 2 3 4 5 6

   ]3                    Identity function
3
```

Exercises

1. What are the commonly-used names for the functions (or verbs) denoted here by `+ * -`
   [plus times minus or addition multiplication (or product) subtraction]
2. Enter `plus=:+` to assign the name `plus` to the addition function, and then experiment with the following expressions:

```
   3 plus 4 * 2
11
   zero=:0
   one=:1
   two=:2
   three=:3
   four=:4
   times=:*
   three plus four times two
```

3. As illustrated by the preceding exercise, much math could be expressed in English words without forcing students to learn the "difficult" special notation of math. Would you prefer to stick to English words?
4. Experiment with the following editing facilities for correcting errors:
   - Correct a line being entered by using the cursor keys (marked with arrows) to move the cursor to any point, and then type or erase (using the delete or backspace keys). The cursor need not be returned to the end of the line before entering the line.
   - Revise any line by moving the cursor up to it and pressing enter to bring it down to the input area for editing.

Not only is it important to think through the expected result of an experiment before executing it on the computer, but it is also a good practice to look for patterns in any lists or tables you may see. Then verify your observations by doing calculations by hand for short lists, and then test them more thoroughly on the computer. For example, the list of odd numbers:

```
   1+2*a
1 3 5 7 9 1
```

may be added by hand to give `36`. Now add only the first five of the list, the first four, and so on down to the first one.

Do you see a pattern in these results? If not, compare them with the following list of squares:

```
   (1+a)*(1+a)
1 4 9 16 25 36
```

It appears that for any value of **n**, the sum of the first **n** odd numbers is simply the square of **n**. This may be tested further as follows:

```
   n=:20
   a=:i.n
   odds=:1+2*a
   odds
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39

   sum=:+/
   sum odds
400

   n*n
400
```

The sum function **+/** gives the sum of its arguments, but calculation of the subtotals (the sum of the first one, the first two, etc.) would provide a more thorough test. Thus:

```
   sum\ odds
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361
400

   (1+a)*(1+a)
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361
400
```

Hereafter we will suggest many experiments without showing the results, expecting students to use the computer to produce them.

B. Ramble or Research

The main point of this book is to introduce a new tool for exploring math, and to foster its use by applying it to a variety of topics. In other words, it provides a ramble through a variety of topics rather than a systematic study of any one of them.

Rambles through any subject can be much more rewarding, and more self-directed, if one has a systematic knowledge of at least some aspect of it. For example, amateur shell-collecting is more interesting to one with some knowledge of molluscs and their classification; walks through parks are more rewarding to one with some systematic knowledge of plant, animal, or insect life; and walks through hills and mountains are made more interesting by a knowledge of elementary geology.

However, any book on rambling would surely fail if stuffed with serious digressions on the systematic study of each interesting point as it is discovered. It is better to provide the reader with effective but unobtrusive pointers to other sources.

Books 2 and 3 provide deeper studies of two branches of math: *arithmetic* and *calculus*. Being that branch of math that deals with whole numbers, arithmetic is the most elementary and accessible of subjects in math but, as treated in Book 2,

it also provides simple introductions to many more advanced topics, including proofs, permutations, polynomials, logic, and sets.
These books are easy to consult because they use the same J notation. Moreover, they incorporate more systematic introductions and discussions of the notation itself. Further texts of this character include Reiter's *Fractals, Visualization and J* [1], and *Concrete Math Companion* [2].

On the other hand, treatments in conventional notation of a wide variety of topics are more readily available in libraries. Use of them in conjunction with the present text will require sometimes difficult translations between J and conventional notation. However, the effort of translation is often richly repaid (as it is in translating from one natural language to another) by deeper understanding of the matters under discussion.

In fact, a deep appreciation of the method of exploration proposed here may best be found in an attempt to write a companion volume to some chosen conventional text. Some guidance in such an endeavour is provided by *Concrete Math Companion* [2], published as a companion to *Concrete Mathematics* [3].

Chapter 2

What Is Math

math is the short form of *mathematics*, for which the British use *maths*, preserving the ugly plural form for a singular noun.

A. Relations

It is commonly thought that math is about numbers. So it is, but numbers are not the only, nor even the most important, concern of math. It would be more accurate to say that math is concerned with *relations*, and with *proofs* of relations.

Although the first chapter dealt only with numbers, it should be clear that the interesting aspects were the *relations* between results. For example:

```
   a=:i.6                    The first six non-negative integers
   b=:?.~6                   The integers in random order
   b
5 1 2 4 3 0

   3*a
0 3 6 9 12 15

   a+a+a
0 3 6 9 12 15

   (3*a)=(a+a+a)             The relation between multiplication and
addition
1 1 1 1 1 1

   a=b                       The lists a and b are not equal
0 1 1 0 0 0

   sort=:/:~
   sort b
0 1 2 3 4 5

   sort a
0 1 2 3 4 5

   (sort a)=(sort b)         But are similar; one is a permutation of the other
1 1 1 1 1 1
```

We will further illustrate this matter of relations by examples that do not concern numbers. For example, the word `'POST'` is said to be an *anagram* of the word `'SPOT'` because the letters of `'SPOT'` can be permuted to give the word `'POST'`. Thus `'SPOT'` and `'POST'` are *similar* in the sense already defined for lists. The similarity of these words may be tested as follows:

```
   w←'SPOT'
   x←'POST'
   sort w
OPST

   sort x
OPST

   (sort w)=(sort x)
1 1 1 1
```

Sorting `w` produces `OPST`. Is it an anagram? We will say that it is, although it is not an English word.

You could (and should) attempt to write down *all* distinct anagrams of `'SPOT'`, finding a surprising number of English words among them. However, this might be rather difficult to do; in a long list of words it is easy to overlook repetitions, and you may not even know how many anagrams to expect all together.

We will now use the anagram function `A.` for this purpose. Its left argument chooses one of many permutations to apply to the list right argument. Thus:

```
   w
SPOT
   8 A. w
POST

   12 A. 8 A. w          The permutation 12 A. is the inverse of 8 A.
SPOT

   0 1 2 3 4 5 6 7 8 A. w
SPOT
SPTO
SOPT
SOTP
STPO
STOP
PSOT
PSTO
POST
   30 A. w
|index error
|   30     A.w
```

The last result shows that there is a limit to the valid left argument; properly so, since there is a limit to the number of different permutations of a list. But how many are there? In the case of a two-item list `'AB'` there are clearly only two

possibilities, the *identity* permutation that leaves the list unchanged, and the one that gives `'BA'`. Thus:

```
   0 1 A. 'AB'
AB
BA
```

Write down all permutations of the list `'ABC'` to convince yourself that there are six possible permutations. Thus:

```
   (i.6)A.'ABC'
ABC
ACB
BAC
BCA
CAB
CBA
```

Exercises

1. Produce all anagrams of various three-letter English words to find those words that have the largest number of proper English words among their anagrams.
2. Did you find any word more prolific than `'APT'`?
3. Find all English words among the anagrams of `'SPOT'`.

In solving the last exercise above, it was necessary to find the largest left argument of `A.` permitted. This could be done by experiment. Thus:

```
   22 A. 'SPOT'
TOSP
   23 A. 'SPOT'
TOPS
   24 A. 'SPOT'
|index error
|   24     A.'SPOT'

   (i.24)A.'SPOT'
SPOT
SPTO
SOPT
SOTP
STOP
STOP
PSOT
PSTO
POST
POTS
PTSO
PTOS
OSPT
OSTP
OPST
OPTS
```

```
OTSP
OTPS
TSPO
TSOP
TPSO
TPOS
TOSP
TOPS
```

But what is the general relation between the number of permutations and the number of items in the list to be permuted? Although we are dealing with English words and anagrams rather than with numbers, this is a proper *mathematical* question because it concerns *relations*. The question can be answered in the following steps:

In a four-letter word, the first position in an anagram can be filled in any one of four ways.

Having filled the first position, the next can be filled from the remaining three letters in three different ways.

The next position can be filled in two ways.

The last position can be filled in one way.

The total number of ways is the product of these, that is, four times three times two times one.

This product over all integers up to a certain limit (**4** in the present example) is so useful that it is given its own name (factorial) and symbol (**!**). Thus:

```
   !4
24
   4*3*2*1
24
   !0 1 2 3 4 5 6 7
1 1 2 6 24 120 720 5040
```

The number of items in a list is a function that is also provided with a symbol:

```
   w3=:'APT'
   #w3
3
   i.!#w3
0 1 2 3 4 5
   (i.!#w3)A.w3
APT
ATP
PAT
PTA
TAP
TPA
```

4. Comment on the following experiments:

```
sort=:/:~
w=:'SPOT'
sort w
table=:(i.!#w)A. w
# table   sort table
```

5. A table with more rows than columns may be displayed more compactly by transposing it. Try the following:

```
transpose=:|:
transpose table
```

The function `A.` applies to lists of numbers as well as to lists of letters (words), and when applied to lists such as `i.3` and `i.4` produces tables that show its behaviour more clearly. The following experiment uses the *link* function (`;`) to box tables and link them together for more convenient comparison:

```
   i=:i.24
   (i A. 'SPOT');(i A. 'ABCD');(i A. 0 1 2 3)
+----+----+-------+
|SPOT|ABCD|0 1 2 3|
|SPTO|ABDC|0 1 3 2|
|SOPT|ACBD|0 2 1 3|
|SOTP|ACDB|0 2 3 1|
|STPO|ADBC|0 3 1 2|
|STOP|ADCB|0 3 2 1|
|PSOT|BACD|1 0 2 3|
|PSTO|BADC|1 0 3 2|
|POST|BCAD|1 2 0 3|
|POTS|BCDA|1 2 3 0|
|PTSO|BDAC|1 3 0 2|
|PTOS|BDCA|1 3 2 0|
|OSPT|CABD|2 0 1 3|
|OSTP|CADB|2 0 3 1|
|OPST|CBAD|2 1 0 3|
|OPTS|CBDA|2 1 3 0|
|OTSP|CDAB|2 3 0 1|
|OTPS|CDBA|2 3 1 0|
|TSPO|DABC|3 0 1 2|
|TSOP|DACB|3 0 2 1|
|TPSO|DBAC|3 1 0 2|
|TPOS|DBCA|3 1 2 0|
|TOSP|DCAB|3 2 0 1|
|TOPS|DCBA|3 2 1 0|
+----+----+-------+
```

B. Proofs

Although proofs are an important (and many would say the *essential*) part of mathematics, we will spend little time on them in this book.

In introducing his book *Proofs and Refutations: The Logic of Mathematical Discovery* [4], Imre Lakatos makes the following point:

> Its modest aim is to elaborate the point that informal, quasi-empirical, mathematics does not grow through a monotonous increase of the number of indubitably established theorems but

> through the incessant improvement of guesses [Italics added] by speculation and criticism, by the logic of proofs and refutations.

The main point of the present book is to exploit a new tool for the exploration of relations and patterns that can be used by both mathematicians and laymen to find those *guesses* that are amenable to, and worthy of, proof. We will defer further discussion of proofs to Chapter 9, partly to allow the reader to garner guesses that can be used to illuminate the discussion.

We will, however, recommend the reading of Lakatos at any point. The book is highly entertaining, instructive, and readable by any layman with the patience to look up the meanings of a small number of words such as *polyhedron*, *polygon*, and *convex*.

The following quotes from Lakatos reflect his view of the importance of guessing:

> Just send me the thereoms, then I shall find the proofs.
>
> Chrysippus

> I have had my results for a long time, but I do not yet know how I am to arrive at them.
>
> Gauss

> If only I had the theorems! Then I should find the proofs easily enough.
>
> Riemann

> I hope that now all of you see that proofs, even though they may not prove, certainly do help to *im*prove our conjecture.
>
> Lakatos

> On the other hand those who, because of the usual deductive presentation of mathematics, come to believe that the path of discovery is from axioms and/or definitions to proofs and theorems, may completely forget about the possibility and importance of naive guessing.
>
> Lakatos

Exercises

6. Read the three pages of Section C, Chapter 5, of Book 2.

C. Summary

In brief, we will interpret math in the following sense: it concerns relations, and provides languages for expressing them, as well as for expressing transformations on tangible representations.

For example, the first four counting numbers can be represented by the list of symbols

```
1 2 3 4:
```

```
   ! 1 2 3 4                          A transformation (or function)
1 2 6 24
```

```
   */\ 1 2 3 4
1 2 6 24
```

A second transformation

```
   (! 1 2 3 4)=(*/\ 1 2 3 4)
1 1 1 1
```

Equivalent to the first

Chapter 3

Function Tables

The pleasures of the table
belong to all ages
Jean Anthelme Brillat-Savarin

and make it plain upon tables
that he may run that readeth it
Habakkuk

The effect of multiplication can be shown rather neatly in a succession of products of a list as follows:

```
   a=: i.6
   0*a
0 0 0 0 0 0
   1*a
0 1 2 3 4 5
   2*a
0 2 4 6 8 10
```

However, a more perspicuous table of products with each item of `a` can be prepared as follows:

```
   a*/a
0 0  0  0  0  0
0 1  2  3  4  5
0 2  4  6  8 10
0 3  6  9 12 15
0 4  8 12 16 20
0 5 10 15 20 25
```

Similar tables can be prepared for other known functions. For example:

```
   (a*/a);(a+/a);(a-/a)
+---------------+------------+----------------+
|0 0  0  0  0  0|0 1 2 3 4  5|0 _1 _2 _3 _4 _5|
|0 1  2  3  4  5|1 2 3 4 5  6|1  0 _1 _2 _3 _4|
|0 2  4  6  8 10|2 3 4 5 6  7|2  1  0 _1 _2 _3|
|0 3  6  9 12 15|3 4 5 6 7  8|3  2  1  0 _1 _2|
|0 4  8 12 16 20|4 5 6 7 8  9|4  3  2  1  0 _1|
|0 5 10 15 20 25|5 6 7 8 9 10|5  4  3  2  1  0|
+---------------+------------+----------------+
```

Much can be learned from such tables. For example, the multiplication table is *symmetric*, that is, each row is the same as the corresponding column, and its transpose (`|:a*/a`) is the same as the table `a*/a` itself. This implies that the arguments of multiplication may be exchanged without changing the product, or, as we say, multiplication is *commutative*. The same may be said of addition, but not of subtraction, which is non-commutative, as is obvious from its table. Tables for both negative and positive arguments are even more interesting. For example, try each of the three tables with the following symmetric argument:

```
   i: 6
_6 _5 _4 _3 _2 _1 0 1 2 3 4 5 6
```

Note how the multiplication table is broken into quadrants of exclusively positive or exclusively negative numbers by the row and column of zeros, and try to explain why this occurs.

The symbol `/` in the sentence `a*/a` denotes an *adverb*, because it applies to the verb `*` to produce a related verb (that is in turn used to produce a table).

It is much easier to interpret a table if it is bordered by its arguments. We will use a second adverb called `table` for this purpose. For example:

```
   b=:2 3 5 7 11
```

```
   a *table b
+-+--------------+
| | 2  3  5  7 11|
+-+--------------+
|0| 0  0  0  0  0|
|1| 2  3  5  7 11|
|2| 4  6 10 14 22|
|3| 6  9 15 21 33|
|4| 8 12 20 28 44|
|5|10 15 25 35 55|
+-+--------------+
```

Bordered multiplication table

```
   +table~ a
+-+------------+
| |0 1 2 3 4  5|
+-+------------+
|0|0 1 2 3 4  5|
|1|1 2 3 4 5  6|
|2|2 3 4 5 6  7|
|3|3 4 5 6 7  8|
|4|4 5 6 7 8  9|
|5|5 6 7 8 9 10|
+-+------------+
```

Bordered addition table

```
   *table~ i:6
+--+-----------------------------------------------+
|  | _6  _5  _4  _3  _2 _1 0  1   2   3   4   5   6|
+--+-----------------------------------------------+
|_6| 36  30  24  18  12  6 0 _6 _12 _18 _24 _30 _36|
|_5| 30  25  20  15  10  5 0 _5 _10 _15 _20 _25 _30|
|_4| 24  20  16  12   8  4 0 _4  _8 _12 _16 _20 _24|
|_3| 18  15  12   9   6  3 0 _3  _6  _9 _12 _15 _18|
|_2| 12  10   8   6   4  2 0 _2  _4  _6  _8 _10 _12|
|_1|  6   5   4   3   2  1 0 _1  _2  _3  _4  _5  _6|
| 0|  0   0   0   0   0  0 0  0   0   0   0   0   0|
```

```
| 1| _6  _5  _4  _3  _2 _1 0  1   2   3   4   5   6|
| 2|_12 _10  _8  _6  _4 _2 0  2   4   6   8  10  12|
| 3|_18 _15 _12  _9  _6 _3 0  3   6   9  12  15  18|
| 4|_24 _20 _16 _12  _8 _4 0  4   8  12  16  20  24|
| 5|_30 _25 _20 _15 _10 _5 0  5  10  15  20  25  30|
| 6|_36 _30 _24 _18 _12 _6 0  6  12  18  24  30  36|
+--+-----------------------------------------------+
```

Tables also provide an interesting and effective way to explore unfamiliar functions. Often, the display of a bordered function table provides a precise and easily-remembered definition of the function. For example:

Relation

```
   <table~ i:5
+--+--------------------------+
|  |_5 _4 _3 _2 _1 0 1 2 3 4 5|
+--+--------------------------+
|_5| 0  1  1  1  1 1 1 1 1 1 1|
|_4| 0  0  1  1  1 1 1 1 1 1 1|
|_3| 0  0  0  1  1 1 1 1 1 1 1|
|_2| 0  0  0  0  1 1 1 1 1 1 1|
|_1| 0  0  0  0  0 1 1 1 1 1 1|
| 0| 0  0  0  0  0 0 1 1 1 1 1|
| 1| 0  0  0  0  0 0 0 1 1 1 1|
| 2| 0  0  0  0  0 0 0 0 1 1 1|
| 3| 0  0  0  0  0 0 0 0 0 1 1|
| 4| 0  0  0  0  0 0 0 0 0 0 1|
| 5| 0  0  0  0  0 0 0 0 0 0 0|
+--+--------------------------+
```

Relations

```
   (<table~ a),.(=table~ a),.(>table~ a)
+-+-----------+-+-----------+-+-----------+
| |0 1 2 3 4 5| |0 1 2 3 4 5| |0 1 2 3 4 5|
+-+-----------+-+-----------+-+-----------+
|0|0 1 1 1 1 1|0|1 0 0 0 0 0|0|0 0 0 0 0 0|
|1|0 0 1 1 1 1|1|0 1 0 0 0 0|1|1 0 0 0 0 0|
|2|0 0 0 1 1 1|2|0 0 1 0 0 0|2|1 1 0 0 0 0|
|3|0 0 0 0 1 1|3|0 0 0 1 0 0|3|1 1 1 0 0 0|
|4|0 0 0 0 0 1|4|0 0 0 0 1 0|4|1 1 1 1 0 0|
|5|0 0 0 0 0 0|5|0 0 0 0 0 1|5|1 1 1 1 1 0|
+-+-----------+-+-----------+-+-----------+
```

Power and “outof”

```
   (^table~ a),.(!table~ a)
+-+-------------------+-+------------+
| |0 1  2   3   4    5| |0 1 2 3 4  5|
+-+-------------------+-+------------+
|0|1 0  0   0   0    0|0|1 1 1 1 1  1|
|1|1 1  1   1   1    1|1|0 1 2 3 4  5|
|2|1 2  4   8  16   32|2|0 0 1 3 6 10|
|3|1 3  9  27  81  243|3|0 0 0 1 4 10|
|4|1 4 16  64 256 1024|4|0 0 0 0 1  5|
|5|1 5 25 125 625 3125|5|0 0 0 0 0  1|
+-+-------------------+-+------------+
```

Roots

```
   %: table~ a
+-+-----------------------------------+
| |0 1       2       3       4       5|
+-+-----------------------------------+
```

```
|0|0 1       _       _       _       _|
|1|0 1       2       3       4       5|
|2|0 1 1.41421 1.73205       2 2.23607|
|3|0 1 1.25992 1.44225  1.5874 1.70998|
|4|0 1 1.18921 1.31607 1.41421 1.49535|
|5|0 1  1.1487 1.24573 1.31951 1.37973|
+-+-----------------------------------+
```

Exercises

1. Produce and examine bordered tables for the following functions:

   `<.  >.`

   `<:  >:`

   `%`

2. Produce and examine bordered tables for the following "commuted" functions:

   `<.~  >.~`
   `<:~  >:~`
   `%~`

3. Produce and examine bordered tables for the following Greatest Common Divisor and Least Common Multiple functions:

   `+.  *.`

   In particular, apply them to the argument `0 1` (as in `+.table 0 1`) and note that with the interpretation of "true" for `1` and "false" for `0` (as was done by Boole), they then represent the logical functions "or" and "and"

4. Explain the equality denoted by the following sentence:

   `(e>:/e)=(e>/e)+.(e=/e=:s 6)`

5. First enter:

   ```
   at=:+/~ e
   mt=:*/~ e
   st=:-/~ e
   dt=:%/~ e
   trans=:|:
   ```

Then comment on the results of the following:

```
at-:trans at

mt-trans mt

st+trans st

dt*trans dt
```

The following exercises suggest a sequence of experiments that should be tried only after reviewing the tips on explorations given in Chapter 1:

Exercises

6. `a=:i.6`
   `+:a`
   `-:a`
   `(+:a)-(-:a)` Double minus half
   `(+:--:)a`

```
Dmh=:+:--:
Dmh a
```

7. Contrast the result of the following sentence with those of Exercise 6:
```
+:--:a
```

8.
```
   +/%#)a
Average=:+/%#
Average a
Average 3 1 4 1 6
```

9. Re-enter the sentence `(a*/a);(a+/a);(a-/a)` from the beginning of this chapter, and compare the result with the following:

```
a(*/;+/;-/)a
f=:*/;+/;-/
a f a
f~ a
```

Chapter 4

Grammar And Spelling

The level is low
but it has not fallen
Jacques Barzun

I can spell all the words that I use
and my grammar's as good as my neighbour's
W.S. Gilbert

A. Introduction

We have already made significant use of J, why trouble us now with its grammar? On the other hand, if grammar is important, why was it not treated first?

In learning our native language we spend years at it and become quite proficient before we even hear of grammar. However, grammar becomes important for more advanced use of the language in clear writing and speaking. Moreover, the teaching of grammar relies on many examples of the use of the language that would not be familiar to a beginner.

Similarly, more advanced and independent writing in J will require knowledge of its grammar. Moreover, we will find it helpful to refer to sentences from earlier chapters to illustrate and motivate discussions of the grammar.

In learning a second language a student has the advantage of already appreciating the purposes and value of language, as well as some knowledge of grammar from her native tongue. On the other hand, one may be seriously misled by such knowledge, and the student is sometimes best advised to forget her native language as much as possible: one may know too many things that are not true.

The beginner in J will already know much of two relevant languages: English, and Mathematical Notation (to be referred to as MN). The knowledge of English grammar is very helpful, especially when we recognize certain analogies between:

- English verbs (action words) and functions such as `+` and `-` and `*`
- Nouns on which verbs act, and the arguments (such as `3` and `4` and `'STOP'`) to which functions apply
- Pronouns such as `a` and `b` and `mt` used in the preceding chapter, and pronouns such as "it" and "she" used in English
- Adverbs (such as `table` in the preceding chapter) that apply to verbs (functions) to produce different, but related, verbs

Knowledge of MN can be very helpful, particularly in providing familiarity with numbers and symbols for common functions, and with some of the purposes of math. On the other hand, MN can be very misleading because it shows little concern for simple and consistent grammar. For example:

- The simple forms **a+b** and **a*b** used for some functions of two arguments is abandoned in others, as in x^n for the **x^n** used in J, and in $\binom{n}{m}$ for **m!n** (the number of ways of choosing **m** things from **n**)
- The rule that a function of one argument precedes its argument (as in **-b** and **sqrt b**) is abandoned in the case of the factorial (**n!**). In J this is written as **!n**.
- The *ambivalent* use of the minus sign to denote two different functions as determined by the number of arguments provided (*subtraction* in **a-b**, and *negation* in **-b**) is not extended to all functions as it is in J. For example, **a%b** and **%b** denote *divided by* and *reciprocal*; **a^b** and **^b** denote *power* and *exponential*; and **a+/b** and **+/b** denote the *addition table* and *sum over*.
- The imposition of hierarchical rules of execution for certain functions: power is performed before multiplication and division, which are performed before addition and subtraction. The reasons for the development of such rules in MN lie in the expressions used for polynomials, and will be discussed further in the corresponding chapter.

B. The Use of Grammar

The rules of grammar determine how a sentence is to be *parsed*, that is, the order in which its parts are to be *interpreted* or *executed*. In particular, these rules cover the use of punctuation, which can make an enormous difference, as illustrated by the following sentences:

The teacher said George was stupid
The teacher, said George, was stupid

The punctuation in J is provided by parentheses, as illustrated by the following sentences from Chapter 2:

```
   a=:i.6
   b=:?.~6
   (3*a)=(a+a+a)
1 1 1 1 1 1

   3*a=a+a+a          Removal of the punctuation yields a quite different result
3 0 0 0 0 0
```

The parsing of a sentence does not depend on the particular word used, but only on the class to which it belongs. Thus the English examples used above would be parsed without change if the nouns *farmer* and *Mary* were substituted for the nouns *teacher* and *George*. Similarly, the sentence **(3*b)=(b+b+b)** would parse the same as **(3*a)=(a+a+a)**.

The classes concerned are called the *parts of speech*. J has only six parts of speech (including the punctuation provided by parentheses), all of which have

been used in earlier chapters. For example, the *nouns* `3` and `2`, and the *verbs* `+` and `*` and `-` occur in the first three sentences in Chapter 1, and the *copula* `=:` (analogous to the copulas *is* and *are* in English) occurs in the next.

As in English, an adverb applies to a verb to produce a related verb. Examples occurring in Chapter 1 are:

> The adverb `/` which inserts its argument function between items of the noun to which it applies. For example, `+/1 2 3 4` is equivalent to `1+2+3+4`, and the function `+/` may therefore be called the *sum* function.
>
> The adverb `\` which uses its argument function to *scan* all prefixes of its noun argument: `+/\1 2 3` is equivalent to `(+/1),(+/1 2),(+/1 2 3)`.

In English, the phrase "run and hide" uses the copulative conjunction "and" to produce a new verb that is a composition of the actions described by the verbs "run" and "hide". In J, `@:` is a conjunction that applies its first argument verb to the result of its second argument verb. For example:

```
   a
0 1 2 3 4 5

   b
5 1 2 4 3 0

   a-b
_5 0 0 _1 1 5

   +/a-b
0
   a +/@:- b
0
   sumdif=:+/@:-
   1 2 3 4 5 sumdif 2 3 5 7 11
_13
```

Exercises

1. Search earlier chapters for further examples of the various parts of speech.
2. State the effect of the adverb `~` in the sentences `a-~b` and `a^~b`.

C. Punctuation and Other Rules

In J, a sentence can be completely punctuated so that the only grammatical rule needed to parse it concerns the use of parentheses. For example, the area of a rectangular field can be computed as follows:

```
   Length=:8
   Width=:6
   Area=:Length*Width
   Area
48
```

If instead the width and the length of the roll of wire available to enclose the field are given, the area may be computed as follows:

```
Roll=:32
Sides=:Roll-(Width+Width)      Extent available for other two sides
Length=:Sides%2
Length*Width                   Area for given roll and width
60
```

The whole may be re-expressed as a single sentence punctuated as follows:

```
Area=:((Roll-(Width+Width))%2)*Width
```

Although long names such as **`Width`** and **`Roll`** can be helpful in understanding the point of a sentence, they can also obscure its structure. Briefer (but still mnemonic) names may be substituted:

```
W=:Width
P=:Roll                        An abbreviation for the perimeter of the field
A=:((P-(W+W))%2)*W
```

Other grammatical rules make it possible to omit some parentheses. The next rule (after the rule for parentheses) is:

- A sentence is executed from right to left

Consequently, the phrase **`(P-(W+W))`** may be re-written as **`(P-W+W)`**. Hence:

```
A=:((P-W+W)%2)*W
```

This can be further simplified by using the fact that multiplication is commutative:

```
A=:W*((P-W+W)%2)
A=:W*(P-W+W)%2
```

Since division is not commutative, this trick cannot be repeated, but because division by two is equal to multiplication by one-half, we have:

```
A=:W*(P-W+W)*0.5
A=:W*0.5*(P-W+W)
A=:W*0.5*P-W+W
```

Although an unparenthesized sentence or phrase is *executed* from right to left, it is easily *read* from left to right. To illustrate this we will use the right-to-left execution rules to fully parenthesize the last sentence above:

```
A=:(W*(0.5*(P-(W+W))))
```

This can now be read from left to right as follows: **`A`** is **`W`** times the value of the entire phrase that follows it, which in turn is **`0.5`** times the phrase that follows *it*, and so on.

The foregoing example made no use of adverbs and conjunctions, and for a sentence that does include them we need a further rule:

- Adverbs and conjunctions are applied before verbs.

For example:

```
+/a*b      is equivalent to    (+/)a*b
^&3 a+b    is equivalent to    (^&3)a+b
```

A complete formal statement of the grammar of J may be found in *J Dictionary* [5], which is also available on the computer by using the Help menu. This statement of the grammar should perhaps be studied at some point, but it is probably better to begin by reviewing familiar sentences and trying to apply the grammatical rules to them. You might review the sentences of earlier chapters as follows:

- Modify and simplify them, using the methods suggested in the foregoing examples (as well as any others that occur to you).
- Try to read the resulting sentences from left to right, using English to paraphrase them.
- Assign values to any names used in the sentences so that they may be entered for execution. If you modify a sentence in any way that changes its meaning, you will probably be alerted to the fact by seeing a different result upon entering it.

The following Exercises highlight points that you might well miss in your review.

Exercises

3. Comment on the sentence `a=:0,1,2,3,4,5` used in Chapter 1 to introduce the first example of a list.

   [The comma denotes a catenate verb that appends one list (or a single item) to another. Also experiment with other forms of catenate as in:

```
b=:i.-6
a,b
a,.b      Called stitch
a,:b      Called laminate
a;b       Called link  ]
```

4. Why is it possible to enter a list of numbers as in `a=:0 1 2 3 4 5` as well as by using the catenate function as in Exercise 3?
   [Certain results that can be produced by functions can also be entered more simply as constants. For example:

This sentence	is equivalent to	this constant
`3-5`	`_2`	
`3+8%10`	`3.8`	
`3%5`	`3r5`	
`3+j.4`	`3j4`	
`2,3,5,7`	`2 3 5 7`]	

5. Read the first five pages of Part II (Grammar) of *J Dictionary* [5] (also available in Help, as described in Chapter 10).

D. Spelling

The many words in English are each represented by one or more letters from a rather small alphabet. The words (nouns, verbs, etc.) of J are each represented by

one or more characters from an alphabet of letters and other symbols. For example:

```
+ +. +: & i. A.
```

Every word of more than one character ends with a dot or a colon.

Any other sequence beginning with a letter and continuing with letters or digits (but not ending with a dot or colon) is a *name* that may be used with a copula, as in the following examples:

`a=:i.6`	Pronoun
`plus=:+`	Proverb
`g=:/\`	Pro-adverb
`p3=:^&2`	Proverb

The representation of numbers is illustrated by:

`2` and `2.4` and `0.4`

`_2` and `_2.4` and `_0.4`

A decimal point must be preceded by a negative sign or at least one digit. As shown in Exercise 4, an `r` may be used in a number to denote a rational fraction (as in `2r3` for two-thirds), and a list may be represented by a list of numbers (as in `2.3 2r3 4`).

The spelling rules of J determine how words are formed from the string of characters that comprise a sentence. They can be clarified by applying the *word-formation* verb to a (quoted) sentence. For example:

```
   ;: '+/4 3 2 1*/i.6'
+-+-+-------+-+-+--+-+
|+|/|4 3 2 1|*|/|i.|6|
+-+-+-------+-+-+--+-+
```

It should also be noted that redundant spaces may be inserted in a sentence to improve readability, as in `a=: i. 6` instead of `a=:i.6` .

Chapter 5

Reports

Cornelius the centurion,
a man of good report
Acts

A. Introduction

If `a` is a list of twelve monthly receipts for a year, then a quarter-by-month report of the same receipts can be obtained as follows:

```
   ]qm=:4 3$ a=:1 7 4 5 2 0 6 6 9 3 5 8
1 7 4
5 2 0
6 6 9
3 5 8
```

The sum over the quarters is given by:

```
   +/qm
15 20 21
```

A two-year report for constant receipts of `10` can be obtained by:

```
   ten=:2 4 3$10
   ten
10 10 10
10 10 10
10 10 10
10 10 10

10 10 10
10 10 10
10 10 10
10 10 10
```

A more realistic report can be obtained by applying the repeatable random generator to this array:

```
   yqm=:?.ten
   yqm
1 7 4
5 2 0
6 6 9
3 5 8

0 0 5
6 0 3
0 4 6
5 9 8
```

The sums over the years of this report are:

```
   +/yqm
 1  7  9
11  2  3
 6 10 15
 8 14 16
```

Because **yqm** has three categories or *axes*, we call it a rank-3 report or *array*. Its rank-2 cells are the two quarter-by-month tables seen in its display, and its rank-1 cells are the eight rows (arranged, in effect, in a **2** by **4** array).

The sums over the quarters in each year are the sums over the two rank-2 cells, yielding a **2** by **3** array (for the two years and three months in each quarter). Thus:

```
   +/"2 yqm
15 20 21
11 13 22
```

Similarly, the sums over the three months in each quarter are a **2** by **4** array given by:

```
   +/"1 yqm
12 7 21 16
 5 9 10 22
```

Exercises

1. Enter the foregoing expressions, and verify that they reproduce the foregoing results.
2. The function **?.** reproduced the same result because it is a repeatable random number generator. Try the expression **?ten** several times to show that the results do not repeat.
3. Predict and verify the results of **+/"3 yqm** and **+/"0 yqm**.
4. Experiment with the box function, as in **<3 4 5** and **<yqm** and **<"2 yqm** and **<"1 yqm**.
5. The sentence **$yqm** gives the shape of the array **yqm**. Apply **$** to other results such as **+/yqm** and **+/"2 yqm** and **+/"1 yqm**.

6. The function `#` gives the number of items or major cells in its argument. Apply it to various arguments.

The expression `f"k` can be used to apply any function `f` to the rank-`k` cells of its argument. For example, the *mean* or *average* function can be used as follows:

```
   mean=:+/%#
   mean 3 4 5 6
4.5
   (mean;mean"2;mean"1) yqm
+-----------+-------------+--------------------------------+
|0.5 3.5 4.5|             |                                |
|5.5   1 1.5|3.75    5 5.25|      4 2.33333       7 5.33333|
|  3   5 7.5|2.75 3.25  5.5|1.66667       3 3.33333 7.33333|
|  4   7   8|             |                                |
+-----------+-------------+--------------------------------+
```

Exercises

7. Experiment with rank cases of the following functions, and state in English the meanings of the various results:

`|.`	Reverse
`2&|.`	Rotate
`#`	Number of items
`$`	Shape

B. Transposition

Given a year-by-quarter-by-month report `yqm` we may want to see the receipts displayed as a quarter-by-month-by-year report `qmy`. If we refer to the successive axes (or categories) by the indices `0 1 2`, we may say that `qmy` is to be obtained by the transposition `1 2 0` (choosing axis `1`, then axis `2`, then axis `0`). Thus:

```
   qmy=:1 2 0 |: yqm
   qmy;yqm;($qmy);($yqm)
+---+-----+-----+-----+
|1 0|     |     |     |
|7 0|     |     |     |
|4 5|     |     |     |
|   |1 7 4|     |     |
|5 6|5 2 0|     |     |
|2 0|6 6 9|     |     |
|0 3|3 5 8|     |     |
|   |     |4 3 2|2 4 3|
|6 0|0 0 5|     |     |
|6 4|6 0 3|     |     |
|9 6|0 4 6|     |     |
|   |5 9 8|     |     |
|3 5|     |     |     |
|5 9|     |     |     |
|8 8|     |     |     |
+---+-----+-----+-----+

   (mean;mean"2;mean"1) qmy
+---------+--------------+-----------+
|3.75 2.75|      4 1.66667|0.5 3.5 4.5|
```

```
|   5 3.25|2.33333       3|5.5   1 1.5|
|5.25  5.5|      7 3.33333|  3   5 7.5|
|         |5.33333 7.33333|  4   7   8|
+---------+---------------+-----------+
```

Transpositions may also be used on higher-rank arrays, as in the following product-by-year-by-quarter-by-month report:

```
   pyqm=: ?. 2 2 4 3$10
   ypmq=: 1 0 3 2 |: pyqm
   ypmq ([;$@[;];$@])pyqm
```

Boxing of various ranks can also be used to clarify displays:

```
   <"2 ypmq
+-------+-------+
|1 5 6 3|5 4 7 7|
|7 2 6 5|0 7 2 3|
|4 0 9 8|6 9 0 6|
+-------+-------+
|0 6 0 5|7 2 7 6|
|0 0 4 9|9 9 6 8|
|5 3 6 8|3 7 0 2|
+-------+-------+

   <"3 ypmq
+-------+-------+
|1 5 6 3|0 6 0 5|
|7 2 6 5|0 0 4 9|
|4 0 9 8|5 3 6 8|
|       |       |
|5 4 7 7|7 2 7 6|
|0 7 2 3|9 9 6 8|
|6 9 0 6|3 7 0 2|
+-------+-------+
```

Chapter 6

Terminology

If this young man expresses himself in terms too deep for me,
Oh what a singularly deep young man this deep young man must be
W.S. Gilbert

Special terminology used in various branches of knowledge often imposes a serious burden on a beginner. It may sometimes be safely dismissed as pretentious and no better than familiar terms, but serious treatment of a topic may well require finer distinctions than those provided by familiar language. For example, the familiar *average* may sometimes be substituted for *mean* as defined in math and statistics. However, *mean* refers not only to *average* (the *arithmetic* mean), but also to various ways of characterizing a collection by a single number, including the *geometric mean*, *harmonic mean*, and *common mean*.

Similarly, the grammatical terms adopted in J (from English) may seem pretentious to anyone familiar with corresponding terms in math, but they make possible significant distinctions that are not easily made in MN. We illustrate this by a few sentences and the classification of items from them in both J and MN:

```
with=:&
cube=:^ with 3
commute=:~
into=:% commute
pi=:7 into 22
2 into cube a=:i.6
```

		J MN
Noun	`22`	Constant
Pronoun	`pi`	Variable
Verb or Function	`%`	Function or Operator
Proverb	`cube`	
Adverb or Operator	`~`	Operator
Pro-adverb	`commute`	
Conjunction or Operator	`&`	Operator
Pro-conjunction	`with`	
List or Vector	`a`	Vector
Table or Matrix	`a*/a`	Matrix
Report or Array	`a+/a*/a`	Array

In the foregoing, MN makes the same distinction made by *noun* and *pronoun* in J, but uses the terms *constant* and *variable*. The term *variable* may prove somewhat misleading, particularly when used for a pronoun such as **pi** (for the ratio of the circumference to the diameter of a circle), which is not expected to *vary*. The following sentences may be used to clarify the choice of the word *variable*:

```
                    sqr=:*:       The square function in J
(sqr 0)=(0+0)
(sqr 2)=(2+2)
(sqr 0)=(0*0)
(sqr 2)=(2*2)
(sqr 3)=(3*3)
```

Each of these sentences express a "true" relation in the sense that each comparison yields **1**. However, the first pair are true only for the specific arguments **0** and **2**, and for no other. The last three suggest (correctly) that the indicated relation remains true for *any* argument, or, as we say, the argument is allowed to *vary*. This generality is commonly indicated by using a *pronoun* argument, or, as stated in MN, a *variable*:

```
(sqr x)=(x*x)
```

In MN, the term *operator* (or *functional*) is used for both of the cases distinguished in J by *adverb* and *conjunction*. Moreover, in MN the term *operator* is also commonly used to refer to functions.

The terms *list*, *table*, and *report* are used in J with meanings familiar to anyone, whereas the corresponding terms *vector*, *matrix*, and *array* might be known only to specialists. The familiar use of *vector* is as a carrier, as in *disease vector*. It might be thought that a vector "carries" its items, but the actual etymology of the term in math is quite different, although not as bizarre as that of *matrix*.

New terminology should be approached by using dictionaries to learn the etymology of terms, both old and new. For example, a *verb* is defined as a word that (amongst other things) expresses an action; the corresponding word *function* comes from a root meaning "to perform".

Attention to etymology is also rewarding in every-day work. For example, the meaning of *atom* appears clearly in its derivation (a[not] + tem[cut]) as something that could not be cut.

The *American Heritage Dictionary* [6] presents etymology in a particularly revealing manner: all words derived from a given root are listed together in an appendix. This highlights surprising and insightful relations, such as that between *tree* and *true*. As a further example, the root *tem* that occurs in *atom* also occurs in *anatomy*, *microtome*, and *tome*. Incidentally, *tome* does not necessarily mean a big book, but rather one of the volumes "cut" from a book, such as the 24 tomes of the original Oxford English Dictionary.

Lewis Thomas, a noted bio-chemist, explored the pleasure and profit of etymology in his delightful book *et cetera, et cetera.* [7]. It is well worth reading.

Exercises

1. Speculate on the possible relation between the similar-sounding words tree and true. Then look them up in AHD [6] , and consult their common Indo-European root in the appendix.
2. Read the entries in the Indo-European sub-dictionary of AHD for the roots ag, ak, ar, and gene, and look up some of the words derived from them.

Chapter 7

Decimal and Other Number Systems

Sixty-four I hear you cry!
Ask a silly question and
get a silly answer!
Tom Lehrer

A. Introduction

To most people, the decimal representation is so familiar, and so closely identified with "the number itself", that it may be difficult to grasp the notion of representation. For example, what is one to make of the assertion:

> The decimal representation of `365` is `365` ?

We will use lists to clarify the discussion:

> The decimal representation of `365` is `3 6 5`
>
> The octal (base-8) representation of `365` is `5 5 5`

```
   bv=:#.                    The base-value function
   10 bv 3 6 5
365

   8 bv 5 5 5
365
```

The main idea of a *base* or *radix* representation is embodied in the function `#.` which we will now re-express in terms of more familiar functions. Familiarity with decimals should make it clear that the representation `3 6 5` is to be evaluated by multiplying the first item by `100`, the second by `10`, and the third by `1`, and summing the products. Thus:

```
   r=:3 6 5
   w=:100 10 1
   r*w
300 60 5

   +/r*w
365
```

The weights `w` would not be appropriate for a list of other than three items, and the following suggests a more general expression:

```
   y=:1 9 9 6
   base=:10
   #y
4
```

```
   i.-#y
3 2 1 0
```

This is the reversal of the list i.#y

```
   base^i.-#y
1000 100 10 1

   +/y*base^i.-#y
1996

   z=: 3 7 1 4
   +/z*8^i.-#z
1996
```

```
   BV=:+/@:(] * [ ^ i.@:-@:#@])
   10 BV 1 9 9 6
1996

   8 BV 3 7 1 4
1996
```

Equivalent to bv=:#.

We may also define and explore specific cases of the base-value function by combining it with various left arguments:

```
   bv10=:10&#.
   bv8=:8&#.
   bv2=:2&#.
   bv8 z
1996

   bv2 1 0 1
5
```

What function will yield the representation of a given argument? In other words, what are the functions inverse to the functions **b10**, **b8**, and **b2**? The adverb **^:_1** gives the inverse of a function to which it is applied. Thus:

```
   inv=:^:_1
   sqrt=:%:
   sqr=:sqrt inv
   sqrt i.6
0 1 1.41421 1.73205 2 2.23607

   sqr sqrt i.6
0 1 2 3 4 5
   bv8i=:bv8 inv
   bv8i 365 1996
0 5 5 5
3 7 1 4

   bv2 inv 365 1996
0 0 1 0 1 1 0 1 1 0 1
1 1 1 1 1 0 0 1 1 0 0
```

```
   2 #. bv2 inv 365 1996
365 1996
```

We learn to add decimal numbers by adding the items of their representations, and performing "carries" as required. What would the result mean if we did not perform the carries? For example:

```
   bv10i=:bv10 inv
   ]d10=:bv10i 365 1996
0 3 6 5
1 9 9 6

   s10=:+/d10
   s10
1 12 15 11

   10#.s10
2361

   365+1996
2361

   d8=:bv8i 365 1996
   d8
0 5 5 5
3 7 1 4

   8#.+/d8
2361
```

It appears that the sum `+/d10` does indeed represent the correct sum in base-`10`. Why then do we normally perform the carries?

We could perform successive carries on the sum `s10` as follows:

```
   1 12 15 11
   1 12 16 1
   1 13 6 1
   2 3 6 1
```

We first verify that `2 3 6 1` represents the correct sum:

```
   d=:2 3 6 1
   (10#.d),(10#.s10),(365+1996)
2361 2361 2361
```

The reason that the representation `d` is preferred is that its items can be simply written side-by-side to give the normal decimal form, whereas the items of `s10` would give the quite different result `1121511`.

Similar remarks apply to bases other than `10`.

Exercises

1. Perform the carries on the base-`8` sum `+/d8` (that is, `3 12 6 9`)
2. Enter `x=:?.4#1000` to obtain four random integers less than `1000`. Then obtain their base-`10` represetations, sum them, and perform the carries

necessary to obtain a normalized representation. Verify the correctness of the final results.

3. Repeat Exercise 2 for bases other than `10`.
4. The method for adding multi-digit decimal numbers commonly taught requires a sequence of carries interleaved with the additions, whereas the method used here first performs all additions, and then performs the carries. Which is the least error-prone? Which is the easier to re-check by repeating all or part of the process?
5. Give a clear statement (in English) of the "carrying" or "normalization" process commonly taught. Include the case of bases other than `10`, as well as the case where a carry occurs from the leading position (thus increasing the number of items in the list).

As suggested in the last exercise, the hand procedure for normalization can be precisely prescribed in English. Can it also be defined as a (computer-executable) function in J? We begin with a process on a specific argument:

```
   y=:3 4 25
   r=.i.0                 Initialize the result as an empty list

   ci=._1{.y              Current item is last item of argument
   r=.(10|ci),r           Prefix remainder to the result list
   c=.<.ci%10             Compute the carry to the next position
   y=.}:y                 Truncate by dropping the treated item

   ci=.c+_1{.y            Add carry to last item
   r=.(10|ci),r
   c=.<.ci%10
   y=.}:y

   ci=.c+_1{.y
   r=.(10|ci),r
   c=.<.ci%10
   y=.}:y
   r
3 6 5

   (10#.r),(10#.3 4 25)
365 365
```

The last two groups of four steps are identical, a uniformity that was achieved by truncating the argument each time. Complete uniformity would allow the entire process to be stated more compactly (and more generally) as a repetition or *iteration* of a fixed procedure defined by the four steps. It remains to make the first block uniform: initialize the carry to zero, and replace the first line of the block as follows:

```
   r=.i.c=.0
   ci=.c+_1{.y
```

The foregoing process may now be defined as an iteration as follows:

```
   NORM=: 3 : 0
r=.i.c=.0
label_loop.

  if. 0<#y. do.
  ci=.c+_1{.y.
  r=.(10|ci),r
  c=.<.ci%10
  y.=.}:y.
  goto_loop.
end.
r
)
   NORM 3 4 25
3 6 5
```

In the foregoing definition:

- The right argument is denoted by **y.**
- The block to be iterated is delimited by **do.** and **end.**
- Repetition of the block is determined by **if.** followed by a condition
- The result of the function is the result of the last sentence (that is, **r**)
- The entire definition is terminated by a right parenthesis alone on a line

A function that works correctly on the argument that guided its definition may not work in general, and should be thoroughly tested. For example:

```
   NORM 10 11 12
1 2 2

   (10#.NORM 10 11 12),(10#.10 11 12)
122 1122
```

The discrepancy clearly occurs because the carry computed in the final iteration is not zero, and must not be ignored. To rectify this, we make the condition for repetition depend upon a non-zero carry as well as upon a non-empty argument:

```
   NORM=: 3 : 0
r=.i.c=.0
label_loop.
  if. (c~:0)+.(0<#y.) do.
  ci=.c+_1{.y.
  r=.(10|ci),r
  c=.<.ci%10
  y.=.}:y.
  goto_loop.
end.
r
)

   NORM 10 11 12
1 1 2 2
```

```
   NORM 1234 5 6
1 2 3 4 5 6
```

The function may now be generalized to a dyadic definition in which the first argument specifies the base used: each occurrence of `10` is replaced by `x.`, and the line `NORM=: 3 : 0` is replaced by `NORM=: 4 : 0` :

```
   NORM=: 4 : 0
r=.i.c=.0
label_loop.
  if. (c~:0)+.(0<#y.) do.
  ci=.c+_1{.y.
  r=.(x.|ci),r
  c=.<.ci%x.
  y.=.}:y.
  goto_loop.
end.
r
)

   (8 NORM 5 3 21);(10 NORM 10 11 12)
+-----+-------+
|5 5 5|1 1 2 2|
+-----+-------+
```

Finally, it will be convenient to define a function whose dyadic case is `NORM` and whose monadic case is `10&NORM`. Thus:

```
   N=:(10&NORM) : NORM
   (8 N 5 3 21);(N 10 11 12)
+-----+-------+
|5 5 5|1 1 2 2|
+-----+-------+
```

Exercises

6. Although the formal definition of the process carried out by `N` is rather involved, the hand-calculation of it is quick and trivial. Confirm this by performing it on various lists, checking the accuracy of your work by applying the function `10&#.` to each list and its normalized form.
7. The copula `=.` used in the definition of `NORM` differs from the `=:` used elsewhere. Its use localizes the assigned name so that it bears no relation to the same name used outside the definition. Experiment with the distinction by defining a function `GNORM` that is identical to `NORM` except for the use of global assignment (`=:`) and compare the behaviour of the two functions. A name can be erased by using `4!:55`, as in `4!:55 <'c'` .

B. Addition

In the example `d10=: bv10i 365 1966` we have already seen how the decimal representations of two numbers may be added to obtain *a* representation of the sum; we may now obtain a *standard* representation by applying the function `N`. Thus:

```
   d10=: bv10i 365 1966
   d10
0 3 6 5
1 9 9 6

   +/d10
1 12 15 11

   N +/d10
2 3 6 1
```

Exercises

8. Use **bv10i** to compute the table of decimal representations of the list of numbers **a,b,c**, where **a=:365** and **b=:1996** and **c=:29**. From this table compute the standard representation of the sum **a+b+c**.
9. Use **ar=:bv10i a** and **br=:bv10i b** and **cr=:bv10i c** to obtain the decimal representations of the numbers of Exercise 1, and use them in expressions to obtain the standard decimal representation of the sum **b**.

In the table produced in Ex. 8, each of the shorter lists (that is, **3 6 5** and **2 9**) are padded with zeroes on the left, a change that does not change the values of the numbers they represent. In Ex. 9 the representations are not so padded, and the lists of differing lengths cannot be added directly. They may be added as illustrated below:

```
   ar;br;cr
+-----+-------+---+
|3 6 5|1 9 9 6|2 9|
+-----+-------+---+
   bv10&> ar;br;cr
365 1996 29

   bv10i bv10&> ar;br;cr
0 3 6 5
1 9 9 6
0 0 2 9
   N +/ bv10i bv10&> ar;br;cr
2 3 9 0

   a+b+c
2390
```

Padding can also be provided more directly, using the fact that the simple opening of a boxed list pads it, albeit on the wrong side:

```
   >ar;br;cr
3 6 5 0
1 9 9 6
2 9 0 0

   pad=:|."1@:(|.&>)
   pad ar;br;cr
0 3 6 5
```

```
1 9 9 6
0 0 2 9
```

C. Multiplication

The commonly-taught methods for addition and multiplication both interleave carries with other computations: in multiplication, each item of the multiplier is applied to the multiplicand and the carries are propagated to give a list of results which are then added to lists for the other items of the multiplier, producing a further sequence of carries. However, as in addition, the carries can all be segregated in a final normalization. For example:

```
   a=:365 [ b=:1996
   ar=:bv10i a [ br=:bv10i b
   t=:ar*/br

   t
3 27 27 18
6 54 54 36
5 45 45 30
```

This table of products may now be summed to collect those corresponding to the same powers of ten, that is, diagonally as follows:

```
   s=:3,(27+6),(27+54+5),(18+54+45),(36+45),30
   s
3 33 86 117 81 30

   (10#.s),(a*b)
728540 728540
```

This may also be expressed by using the *oblique* adverb **/.**, which applies its function argument to each of the diagonals. Thus:

```
   ]s=:+//.t
3 33 86 117 81 30
```

Exercises

10. Carry out by hand the process defined by **+//.ar*/br** for various values of **ar** and **br**, and test the correctness of the resulting products.
11. Experiment with the expression **</.ar*/br** to get a clear view of the behaviour of the oblique adverb.
12. Define and test a function **TIMES** such that **ar TIMES br** gives the standard decimal representation of the product of numbers whose decimal representations are **ar** and **br**.

A clearer view of the justification for the diagonal sums used in the expression `+//.t` can be obtained by producing a table of powers of ten which multiplied by `t` gives products weighted by the appropriate powers of ten:

```
    a=:365 [ b=:1996
   ar=:bv10i a [ br=:bv10i b
   t=:ar*/br
   ea=:i.-#ar [ eb=:i.-#br
   exp=:ea +/ eb
   wp=:10^exp
   wpt=:t*wp
   wpt
300000 270000 27000 1800
 60000  54000  5400  360
  5000   4500   450   30

   +/+/wpt
728540
   a*b
728540

   TIMES=:N@(+//.@(*/))
   ar TIMES br
7 2 8 5 4 0

   (10#.ar TIMES br),(a*b)
728540 728540
```

Exercises

13. Perform hand-calculations of products using the process defined by the function `TIMES`, and compare its use with the commonly-taught process. Which requires the most writing? Which is the more error-prone? Which is the easier to re-check by re-calculation of parts of the process?

D. Subtraction

Subtraction leads to the question of representing negative arguments. We will use lists of negative numbers, with the standard form limited (as it is for positive arguments) to numbers whose magnitudes are less than the base. For example:

```
   10#. _3 _6 _5
_365

   10#._3 _4 _25
_365
```

The function `bv10i=:10&#.^:_1` can be used to obtain the representation of a negative number by applying it to the magnitude, and then multiplying the resulting list by `_1`. Thus:

```
   c=:_365
   ar=: _1 * bv10i@| a
```

```
   cr
_3 _6 _5
```

A corresponding function for either positive or negative arguments can be obtained by multiplying not by `_1`, but by the signum of the argument:

```
   * 365 0 _365
1 0 _1

   REP10=: * * 10&#.^:_1@|
   REP10 _365
_3 _6 _5

   REP10 365
3 6 5
```

With this representation of negative numbers, expressions for addition apply equally for subtraction. For example:

```
   a=:365
   b=:1996
   t=:REP10 a,b
   t
0 3 6 5
1 9 9 6

   -/t
_1 _6 _3 _1

   (10#.-/t),(a-b)
_1631 _1631
```

The normalization function must be modified in the same manner:

```
   NOR=: *@#. * NORM&|
   N=:10&NOR : NOR
   N 3 4 25
3 6 5

   N _3 _4 _25
_3 _6 _5
```

Exercises

14. Read Chapter 4 of Book 2 (*Arithmetic*), and try some of its Exercises. Note particularly the section on Mixed Bases.

Chapter 8

Recursion

re-, back + *currere*, to run
AHD[5]

The factorial function `!` introduced in Chapter 2 was seen to be a product of the first positive integers. Thus:

```
   !n=:4
24

   (4*!3),(4*3*!2),(4*3*2*!1),(4*3*2*1)
24 24 24 24
```

It would therefore appear that `!n` might be defined simply as `n*!n-1`. Such a definition is said to be *recursive*, because the function being defined *recurs* in its own definition. But a sequence of the form:

```
f n
n*f n-1
n*(n-1)*f n-2
```

would continue forever (through `n=:0` and `n=:_1` etc.), and it is clear that two further pieces of information are required: when to stop the process, and the value of the function for the argument at the stopping point. For the present case of the factorial, the stopping condition could be that the argument be `1`, and the stopping value could be given by the identity function `]`. The three required functions are:

```
   p=:]*f@:<:
   q=:]
   r=:1&=
```

The complete definition may now be expressed and used as follows:

```
   f=:p`q@.r
   f 4
24

   f"0 (1 2 3 4 5)
1 2 6 24 120
```

In the definition of **f**, the conjunction ` *ties* the functions **p** and **q** to form a *gerund*, from which the *agenda* conjunction selects one for execution according to the index (**0** or **1**) provided by its right argument function **r**. Once **f** is defined as above, we can experiment with **p** and the other functions to see some of the workings of the definition of **f**:

```
   p                    Display the definition of p
] * f@:<:

   p 4
24

   r 4
0

   r 4 3 2 1
0 0 0 1

   q 1
1
```

Exercises

1. Compare the results of **f"0(4 3 2 1 0)** and **!4 3 2 1 0** and redefine **f** so that it agrees with **!** for the argument **0**.

The problem of Exercise 1 could be solved by redefining **q** and **r** as follows:

```
   q=:>:
   r=:0&=
   f 0
1
```

However, it would seem more straightforward to define **q** as the constant **1** as follows:

```
   q=:1
   f 0
|domain error
|       f 0
```

A problem arises because **1** is a *noun*, not a *function*, and the arguments in the gerund **p`q** must both be *functions*. We therefore need a *function* that returns the constant value **1** when applied to any argument. Such *constant functions* are commonly needed, and are produced by the rank conjunction (**"**), used in Chapter 5 to modify a function, as in **<"2**. Thus:

```
   1"0 x=:i.4          Rank 0 produces a result for each atom of x
1 1 1 1

   1"_ x               Infinite rank gives a single result for any argument
1

   x"1 'Now is the time'
```

```
0 1 2 3
```

The function `q` may therefore be redefined as follows:

```
   q=:1"_
   f"0 (4 3 2 1 0)
24 6 2 1 1
```

Finally, `f` (of rank `0`) may be redefined compactly as follows:

```
   f=:(]*f@:<:)`(1"_)@.(0&=)"0
   f 4 3 2 1 0
24 6 2 1 1
```

As a second example of recursive definition we will define the sum of the first `n` odd numbers, first met in Chapter 1:

```
   sod=:0"_`(>:@+:@<: + sod@<:)@.*
   sod 4
16

   sod"0 i.6
0 1 4 9 16 25
```

The definition of `sod` may be interpreted as follows: When the argument `n` is `0`, then the signum on the right returns `0`, choosing the leading function in the gerund, giving a result of `0`; otherwise, the result is the `n`th odd number (that is, `>:@+:@<:`) plus the sum for an argument `n-1` (that is, `sod@<:`).

Exercises

2. For convenience, certain constant functions are provided directly, without the need for the rank operator. Experiment with the constant functions `_9:` and `_8:` and so on through `9:`. Use `1:` and `0:` to simplify the definitions of `f` and `sod` above.
3. Because increment (`>:`) is the inverse of decrement (`<:`), the expression `>:@+:@<:` is of the form `gi@f@g`, where `gi` is the inverse of `g`. We say that this is a case of applying `f` under `g`, and denote it by `f&.g`. Use this fact to simplify the definition of `sod`, and check the resulting behaviour.

Recursive definition essentially specifies a function in terms of the same function applied to a simpler case, and its use can enormously simplify many definitions. For example, the Tower of Hanoi puzzle is stated as follows:

> A set of `n` drilled discs of different diameters stacked as a pyramid on a peg A is to be moved one at a time to a peg C without ever placing a larger on a smaller. A third peg B may be used as intermediary.

The process for two discs may be expressed by the table:

```
AB
AC
BC
```

which is to be interpreted row-by-row as follows:

Move (the top) disc from A to B
Move from A to C
Move from B to C

The case of `n` discs can be expressed in terms of the case of one fewer as follows: Move `n-1` discs to the intermediary peg `B`, then move the remaining largest disc to `C`, and finally move the `n-1` discs from `B` to `C`. We will use this fact to make a recursive definition as follows:

```
   H=:m`b@.(1&=@[)
     m=:(<:@[ H 1: A. ]) , b@] , <:@[ H 2: A. ]
     b=:,:@(0 2&{)@]
   p=:'ABC' Pegs
   1 H p
AC

   2 H p
AB
AC
BC

   |: 3 H p Transposed table
AACABBA
CBBCACC
```

Exercises

4. Use discs and pegs (or numbered cards and labelled positions on a table) to carry out the instructions in the foregoing tables to verify that they provide proper solutions to the Hanoi puzzle. Also enter the expression `|: 3 H p` and test it as well.
5. Give an expression for the number of moves required for `n` discs.
6. Explain the behaviour of the definition of `H`, using experiments to show the permutation provided by the function `A.`, the selection provided by the indexing function `{`, and the purpose of the monadic function `,:`. Also redefine the main function `m`, using indexing to perform the necessary permutations.
7. Experiment with the function `HV=: |:@H`.
8. Read the definition of agenda in [5], and experiment with the use of `$:` for self-reference in recursive definitions.
9. Compare the following recursively-defined function `n` with the first definition of `NORM` in the preceding chapter:
   ```
   f=:(0:,10&|) + <.@(%&10) , 0:
   ```

```
g=:+/@(*./\@(0&=)) }. ]
h=:*./@(10&>)
n=:n@f`g@.h
```

Chapter 9

Proofs

Drug thy memories, lest thou learn it,
lest thy heart be put to proof
Tennyson

A. Introduction

It is probably advisable to begin by reviewing the brief discussion of proofs at the end of Chapter 2.

The final experiment of Chapter 1 showed a relation between the sum of the first **n** odd numbers and the square of **n**. We will first reproduce it here:

```
   n=:20
   odds=:1+2*a=:i.n=:20
   odds
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39

   (+/odds),(n*n)
400 400

   +/\odds
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361
400

   (1+a)*(1+a)
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361
400
```

But is the indicated relation true for *any* positive integer **n**? If you are already convinced that it is, any *proof* may seem pointless. However, you might still ask *why* it is true. The following should be helpful in answering this:

```
   q=:1+2*i.n=:6          First six odd numbers
   r=:|.q                 Odds in reverse order
   q,:r
 1 3 5 7 9 11
11 9 7 5 3  1
```

```
   (+/q);(+/r);(q+r);(2%~q+r);(+/2%~q+r)
+--+--+-----------------+-----------+--+
|36|36|12 12 12 12 12 12|6 6 6 6 6 6|36|
+--+--+-----------------+-----------+--+
```

The foregoing shows the rather obvious fact that sums over a list, over the reversed list, and over one-half of the sum of the lists all agree. But the half-sum of the lists has a pattern whose sum is easily expressed as a product:

```
   (2%~q+r);(n#n);(+/n#n);(n*n)
+-----------+-----------+--+--+
|6 6 6 6 6 6|6 6 6 6 6 6|36|36|
+-----------+-----------+--+--+
```

The last agreement (between `+/n#n` and `n*n`) is based on the fact that multiplication is defined as repeated addition.

The foregoing attempted to show why two results were equal by exhibiting their equivalence to other results, where the equivalence was already known or *obvious*. This is perhaps the only way to answer the question *why*. However, the equivalences assumed may be made clearer by laying out the steps of the argument as a *proof*, that is, as a succession of equivalent statements annotated by the justification of the equivalence of each to the one preceding it. Thus:

`+/q=:1+2*i.n`	
`+/|.q`	Summation is symmetric (unaffected by ordering)
`2%~(+/q)+(+/|.q)`	Half sum of equals is an identity
`2%~+/(q+|.q)`	Summation is symmetric
`+/2%~(q+|.q)`	Summation distributes over division
`+/(n#n)`	
`n*n`	The definition of multiplication

Such a list of supposedly equivalent sentences can be tested (for careless errors) by assigning a suitable value to the argument `n`, entering them on the computer, and comparing the results.

This putative proof has not proved anything but it has, as Lakatos would say, broken the original conjecture into a collection of sub-conjectures, each of which may be profitably examined. Consider the first assertion that summation is symmetric, and gives the same result when applied to any permutation of a list. This may be tested as follows:

```
   q=:1+2*i.n=:6
   117 A. q
1 11 9 5 7 3

   _1 A. q
11 9 7 5 3 1

   (+/q),(+/117 A. q),(+/_1 A. q)
36 36 36
```

But *why* is summation symmetric? We may, for example, ask whether the notion applies to other functions, as in product over (`*/`), maximum over (`>./`), and

subtraction over
(`-/`), beginning with the following tests:

```
   r=:|.q
   (+/q),(*/q),(>./q),(-/q)
36 10395 11 _6

   (+/r),(*/r),(>./r),(-/r)
36 10395 11 6
```

What is it about the functions `+`, `*`, and `>.` that make `+/`, `*/`, and `>./` symmetric? The answer is that they are both *associative* and *commutative*. These matters are examined further in Exercises, but the main point is that *any* conjecture may lead to further sub-conjectures that can be identified and pursued until the reader reaches assertions that are satisfying to him. As Lakatos shows, assertions satisfactory for one reader (or purpose) may not be satisfactory for another.

Exercises

1. Addition is said to be associative because a sequence of additions can be associated by parenthesizing them in any way without changing the result. For example, `+/1 2 3 4` and `(1+(2+(3+4)))` and `((1+2)+(3+4))` and `(1+(2+3)+4)` are all equal. Test the associativity of addition by entering a variety of equivalent expressions.
2. Repeat Exercise 1 for product and maximum.
3. The completely parenthesized form of `+/q` is `1+(3+(5+(7+(9+11))))`, and the corresponding form of `+/117 A. q)` is `1+(11+(9+(5+(7+3))))`. Write a sequence of sentences [such as `1+(3+(5+(7+(11+9))))`]that uses only associativity and commutativity to move from the first expression to the last, and enter them all to test their equivalence.
4. Use the words Comm and Assoc to annotate your solution to Exercise 3 to provide a formal proof of the equivalence of `+/q` and `+/117 A. q`.
5. The proof that `+/q` is equivalent to `n*n` is completely formal except for one omission. Complete it.

Following Lakatos's point that a formal or informal proof may suggest further lines of inquiry, we note that the list sum `q+|.q` gave items with a common value. This is, of course, a proposition that is not true for every list `q`, but depends upon some property of `q`. What is that property?

The point is that `q` is an *arithmetic progression*; successive items increase by the addition of a fixed constant (in this case `2`). The sum of the first and last items therefore equals the sum of the item just following the first and just preceding the last, and so on for further pairs. This is more easily stated (and seen) by reversing the list to bring corresponding pairs together. Thus:

```
   q,:|.q
 1  3  5  7  9 11
11  9  7  5  3  1
```

```
   +/q,:|.q
12 12 12 12 12 12
```

The method of proof can therefore be applied to find expressions equal to the sum of any geometric progression. For example:

```
   g=:i.n=:6
   g,:|.g
0 1 2 3 4 5
5 4 3 2 1 0

   +/g,:|.g
5 5 5 5 5 5

   (n*n-1)%2
15

   +/g
15
```

```
   b=:4 [ s=:3 [ n=:7
   h=:b+s*i.n
   h
4 7 10 13 16 19 22

   +/(h,:|.h)%2
13 13 13 13 13 13 13

   b+(s*n-1)%2
13

   n*b+(s*n-1)%2
91

   +/h
91
```

An AP beginning at `b` with steps of size `s`

Exercises

6. Write formal proofs for each of the foregoing results.
7. Define a function `f` such that `f b,s,n` gives the mean of the arithmetic progression beginning at `b` and continuing with increments `s` for a total of `n` items.

B. Inductive Proof

An *inductive* proof of the equivalence of two functions proceeds by first *assuming* that they are equal for some unstated value of the integer argument `n`, and using that assumption (called the *induction hypothesis*) to prove that they are therefore equal for the next argument `n+1`. It is then shown that they are indeed equal for some specific argument `n=:k`. It therefore follows that they are equal for all values `k`, `k+1`, `k+2`, and so on without limit. For example:

```
   ssq=:+/@*:@i.@>:"0
   ssq 5                Sum of squares of first 6 non-negative integers
55

   ssq i.6
0 1 5 14 30 55
```

Using rational constants (such as `2r6` for `2%6`), we then define a putative equivalent function `g`, adopt the induction hypothesis that `f n` is equal to `g n`, and use it to prove that `f n+1` equals `g n+1`:

```
   g=:(1r6&*)+(3r6&*@(^&2))+(2r6&*@(^&3))
   ssq n+1
   +/ *: i. >: n+1                          Definition of ssq
   (+/*:i.>:n)+(*:n+1)                      (Sum of first terms) plus last term
   (ssq n)+(*:n+1)                          Definition of ssq
   (g n)+(*:n+1)                            Induction hypothesis
   (1r6*n)+(3r6*n^2)+(2r6*n^3)+(*:n+1)      Definition of g
   (1r6*n)+(3r6*n^2)+(2r6*n^3)+1+(2*n)+(n^2)
   (1r6*n+1)+(3r6*(n+1)^2)+(2r6*(n+1)^3)
   g n+1                                    Definition of g
```

The lines of the foregoing proof that are not annotated concern the use of manipulations from elementary algebra, including the expansion of the square and the cube of the sum `n+1`. The inductive proof may now be completed by showing that the functions are equal for the argument `0`.

Exercises

8. Enter `n=:6`, and then enter the lines of the foregoing proof to verify that they each give the same result. It is advisable to enter such a sequence in a "text" or "script" file, then execute it, observe the result, and return to the script file to correct any errors and re-try. To open the script file, hold down the control key and press `n`; to execute it, hold down both the control and shift and press `w`; to see the result, switch to the execute window by holding down control and pressing the tab key; return to the script window by the same action.
9. Define the function `s=: +/@:i.@>:` and an equivalent function `t` that does not use summation. Give an inductive proof that they are equivalent.

A recursive definition of a function `f` provides a clear statement of the value of `f n+1` in terms of the value of `f n`; this fact is obviously valuable in the construction of an inductive proof.

But how does one find a function such as `g`? This matter will be treated in Chapter 14. But for present use in further experiments with inductive proofs, we provide the following methods.

The function `g` is an example of a *polynomial*, a sum of weighted powers of the argument, the weights being `0 1r6 3r6 2r6`. They may be obtained as follows:

```
   ]w=: (ssq a) %. a ^/ a=: i.5
```

```
_2.99066e_14 0.1666667 0.5 0.3333333 _6.50591e_14

   6*w
_1.7944e_13 1 3 2 _3.90354e_13
```

Because `%.` (matrix divide) produces its results by approximation, the extreme items of `6*`w are not quite zero. They can be "zeroed" by the following function, in which the first argument specifies the tolerance in number of decimal digits:

```
   ZERO=: ] * |@] > 10&^@-@[
   8 ZERO 6*w
0 1 3 2 0

   14 ZERO 6*w
_1.7944e_13 1 3 2 _3.91687e_13
```

For convenience in experimenting with a variety of functions, we will adopt from Section F of Chapter 14 the conjunction `FIT`, so defined that `n FIT f x` gives the `n`-item list of coefficients of a polynomial that best fits the function `f` at the points `x`. For example:

```
   V=:] ^/ i.@[
   FIT=:2 : 'y. %. (x. & V)'

   3 FIT ^
^ %. 3&V

   ]c=:3 FIT ^ b=:0.2*i.5
1.00238 0.9203119 0.7569838

   c p. b
1.00238 1.21672 1.49162 1.82708 2.2231

   ^ b
1 1.2214 1.49182 1.82212 2.22554
```

As remarked, `g` is an example of a polynomial, and the coefficients produced by `FIT` can (preferably after being zeroed) be used with the polynomial function `p.` to produce an equivalent function. Thus:

```
   ]c=: 8 ZERO 4 FIT ssq a=:i.5
0 0.1666667 0.5 0.3333333 0

   c p. i.8
0 1 5 14 30 55 91 140

   g i. 8
0 1 5 14 30 55 91 140
```

Exercises

10. Study the discussion of proofs in Section D of Chapter 5 of Book 2.

11. Find a function equivalent to the sum of cubes, and construct an inductive proof of the equivalence.

```
]c=: 8 ZERO 5 FIT scubes x=:i.6
```

12. For many functions, the coefficients for an equivalent or approximate polynomial may be conveniently obtained by using the Taylor adverb `t.`, as in `f t. i.6`. Experiment with this for the functions:

```
]             ^&4          (^&4-^&2)

(>:^4:)       (<:^4:)      ^
```

Chapter 10

Tools

Without tools he is nothing,
with tools he is all
Carlyle

A. Introduction

This chapter concerns tools for exploration. They are fully treated in Burke's *J User Manual* (available on-line under the *help* menu in the J system), but should themselves be explored in the manner used for math in preceding chapters.

For example, an overall view of the tools available may be obtained by *dropping* the menus. This can be done by clicking the mouse on each of them, but they can also be dropped by first pressing the *alt* key, then the down arrow, then the left or right arrow to move over the menus. The alt key will roll up a menu.

With a menu dropped, use the up and down arrows to select an item, and press enter to execute it. Alternatively, an underscored letter in an item can be entered to execute it. Some menu items can be invoked directly (without dropping the menu) by pressing a key (usually while holding down the control key), as indicated to the right of the item's name.
For example, (as shown in the *help* menu), the F1 key may be pressed to display the J vocabulary, and any entry in the vocabulary may be chosen for display by double-clicking on it with the mouse. A definition is then displayed, and may also be printed by using *Print topic* in the file menu.

Exercises

1. Using items from the help menu, display and read various pages from the User Manual, including Chapter 1.
2. Display and read a few sections from the introduction to the J dictionary
3. Read the section on grammar in the J dictionary.

B. Editing

As remarked in Chapter 1, a previously entered line can be brought to the input area for editing and re-entry by moving the cursor up to it and pressing *enter*. Moreover, a line containing any phrase can be found by pressing Control f to highlight the search entry box, entering the phrase in it, and pressing *enter*. Repeated searches on the same phrase will find successive occurrences of it.

Pressing Control d drops a menu of previous entries; one may be selected for use by pressing the up arrow.

C. Script Windows

Enter Control n to open a *script window*, enter one or more J sentences in it, and press Control-Shift w to execute the sentences. The execution occurs in the execution window, and can be viewed by entering Control Tab to switch back to it.
A window may be saved as a file (under the name shown on the window) by pressing Control s, and can be re-opened at any time by pressing Control o. It can also be saved under any chosen name by using *Save As* or *Save Copy As* from the file menu.

Exercises

Select the item Session Manager from the User Manual, and from it select the item Script Windows. Read the discussion of their use.

Chapter 11

Coordinates and Visualization

It was their belief that, if they stared long enough at these mystic curves and angles, red ink would turn into black.
Alva Johnson

A. Introduction

Take a sheet of *graph* or *squared* paper (ruled with equidistant vertical and horizontal lines), choose some point of intersection as the *origin* to be labelled `0 0`, and label vertical lines from left to right and horizontal lines from bottom to top with symmetric integers as follows:

```
   i: 9
_9 _8 _7 _6 _5 _4 _3 _2 _1 0 1 2 3 4 5 6 7 8 9
```

Any point of intersection may then be labelled by two coordinates, the first (or **`x`**) coordinate specifying the vertical line through it, and the second (or **`y`**) coordinate the horizontal. Such a coordinate system makes it possible to describe geometric figures, and leads to *analytic* or *coordinate* geometry. For example:

`p=:3 4`	A single point
`q=:9 4`	
`r=:6 8`	
`s=:9 7`	
`t=:8 6.`	
`is=:p,q,:r`	Isosceles triangle
`rt=:p,q,:s`	Right (-angled) triangle
`qd=:p,q,s,:r`	Quadrilateral
`pg=:p,q,s,r,:t`	Pentagon

Properties of the geometric figures can be obtained from their coordinate representations. For example:

`disp=:1&|. - ]`	Rotate by 1 and subtract

```
   disp is                              Displacements from vertex to vertex
 6  0
_3  4
_3 _4

   length=:+/&.*:"1                     Length according to Pythagoras
   length p                             Length or distance from origin
5
   length disp is                       Lengths of sides of isosceles triangle
6 5 5

   heron=:%:@(*/@:(semip,semip-]))      Heron's formula for area
     semip=: 2:%~+/                     Semi-perimeter
   heron length disp rt                 Area of the right triangle
9

   area=:heron@:length@:disp            Area function using Heron
   area rt
9

   area is                              Area of the isosceles triangle
12
```

Exercises

1. Plot the points `p` through `t` on graph paper, and join the appropriate points by straight lines to show the figures `is` through `pg` . Then use the base and altitude of each triangle to compute their areas, and compare with the results of Heron's formula.
2. Use the AHD[6] to examine the etymology of the several terms used for figures that differ only in the number of their sides (or angles or vertices), and suggest a compact common terminology.

   [3-gon, 4-gon, and n-gon (from polygon)]
3. A vertex may be shifted to the left by subtracting a vector with a zero final element. Plot the following triangles, and use both base-times-altitude and Heron's formula to compute their areas:

```
   rts=:p,q,:r-8 0
   is=:p,q,:s-8 0
```

Although plotting polygons by hand may be instructive, it is also convenient to use the computer to plot them. We begin by normalizing the coordinates of a figure:

- sliding them to bring the lowest point to 0 0
- sizing them to no more than 1 in magnitude
- doubling and subtracting 1 to bring them between _1 and 1
- ravelling them to form a list for use by the plotting function

```
   slide=:] -"1 <./
   size=:] %"1 >./
   scale=:,@(<:@+:@size@slide)
   slide is
```

```
0 0
6 0
3 4
   size slide is
  0 0
  1 0
0.5 1
   <: +: size slide is
_1 _1
 1 _1
 0  1
   scale is
_1 _1 1 _1 0 1
```

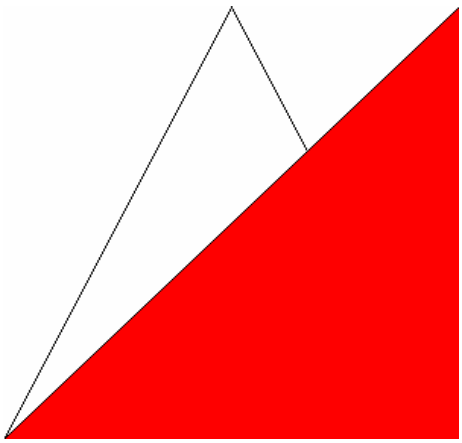
The following steps introduced the necessary graphing functions, and use them to display the isosceles triangle:

```
   load 'graph'
   gdopen'a'
```
Opens graph window labeled 'a'. Use mouse to return focus to J

```
   gdpolygon scale is
   gdshow''
```

We then superpose a red right triangle and, finally, clear the window:

```
   255 0 0 gdpolygon scale rt
   gdshow''
```
Colors red, green, blue; intensity 0-255



```
   gclear 'a'
```

A graphics window may be closed by clicking the upper right corner with the mouse.

The functions provided by the graphics file may be displayed by entering **`names_z_''`** . However, they should for the moment be treated as *tools*, whose internal workings may be ignored provided that their effects are sufficiently understood.

It will be found most convenient to enter a sequence of graphics commands in a script window (opened by entering Control **`n`**), and to execute them by using the drop-down run menu.

To learn more about the use of graphics, use the mouse to drop the Studio menu in J, then click on Labs, and then on Graph Utilities.

Exercises

4. Enter the foregoing sequence of graphics sentences in a script window, and use the "Selection" option from the run menu to execute it.
5. Display each of the polygons defined in this section in various colors; in particular, display **rt** in red and (without clearing the window) **-:is** in green.
6. Permute the coordinates of the polygons (as in **1 A. pg)**, and discuss the appearance of the resulting figures.
7. Enter **rot=:^@j.@rfd@[*]**, and **rfd=:%&180p_1**, and experiment with **rot** by plotting the results of the following forms:

   ```
   45 rot rt

   45&rot&.> rt;is;rts
   ```

8. Experiment with, and comment on, the function **rotate** introduced by the graphics file.

B. Visualization

The examples of Section A illustrate the fact that the coordinate representation and the graphic representation of figures are complementary; each provides certain advantages. For example, the graph of Exercise 6 shows how easy it is to distinguish an "improper" polygon (in which sides cross), a matter that would not be easy to spot in a table of coordinates.

On the other hand, for the computation of properties such as areas, coordinates are far superior. For the particular triangles **rt** and **is** (and even for **rts** and **iss** plotted by hand in Exercise 3) the computation of area appears simple, but this simplicity is deceptive, as illustrated by the rotated figure of **rts** in Exercise 7.

Moreover, the determinant function provides an even simpler statement of area than does Heron's formula, and yields additional important information. Thus:

```
   det=:-/ . *

   rt,"1 (0.5)
3 4 0.5
9 4 0.5
9 7 0.5

   det rt,"1 (0.5)
9

   det (1 A. rt),"1 (0.5)
_9

   AREA=:det@(,"1&0.5)
   AREA rt
9
```

Exercises

9. If you are familiar with the computation of determinants from high school, check the foregoing results by hand.
10. The result of **AREA** is positive if the coordinates are in counter-clockwise order (when plotted), and are negative if clockwise. Test this for various triangles.
11. What is the significance of a zero result from **AREA**?
12. Enter **t=:?.7 2$10** to generate a random table of seven points. Referring to these points by the letters **A** through **G**, determine which of the last five lie on opposite sides of the line determined by the first two.

    [Enter **L=:0 1 { t**, and compare signs of
    the areas of the triangles **C,L** and **D,L**, etc.]

13. Compute the area of the pentagon **pg** of Section A.

    [Referring to the points by **A-E**, compute the three (signed)
    areas **A,B,:C** and **A,C,:D** and **A,D,:E** and add them]

C. Plotting Multiple Figures

As illustrated by Exercise 4, different figures may be displayed together. However, as seen from the same exercise, they are scaled independently, and therefore do not give a satisfactory picture.

We will now rectify this by developing functions that will handle arguments of the form **rt;is;pg**, and scale the whole according to the requirements of the entire collection. It suffices to modify the functions **slide**, **size**, and **scale** so as to apply to each box (that is, *under* (**&.**) *open* (**>**)), and to find the maxima and minima after *razing* the argument (by applying **;**). Thus:

```
SLIDE=:] -"1&.> <@(<./@;)
SIZE=:] %"1&.> <@(>./@;)
SCALE=:,&.>@(<:@+:&.>@SIZE@SLIDE)
```

We may then proceed with experiments such as the following (which plots the isosceles triangle together with the right-triangle displaced two places up and to the right:

```
POLY=:gdpolygon&>
color=:0 255 0;255 0 0
gdopen''
color POLY SCALE is;2+rt
gdshow''
```

Exercises

14. Experiment with the plotting of multiple figures, using expressions of the form:

```
(255 0 0;0 0 255) POLY SCALE rt;pg
```

15. Enter **SCALE <rt** and **SCALE rt** to see that only the former gives the desired result. Define a corresponding function **M** that works in either case

[`M=:SCALE@ bifo=:<^:(-:>)`. Observe the results of `bifo` (box if open) applied to `rt` and to `<rt`.]

16. Enter the definition `reg=:+.@^@j.@o.@(2:*i.%])`, and verify that `reg 4` and `reg 6` give the coordinates of regular polygons inscribed in a unit circle. This definitionemploys complex numbers, so do not spend time on the definition itself at this point. Instead plot the figure `reg 6` and various permutations of it, and interpret the figures observed. Include the following sequence:

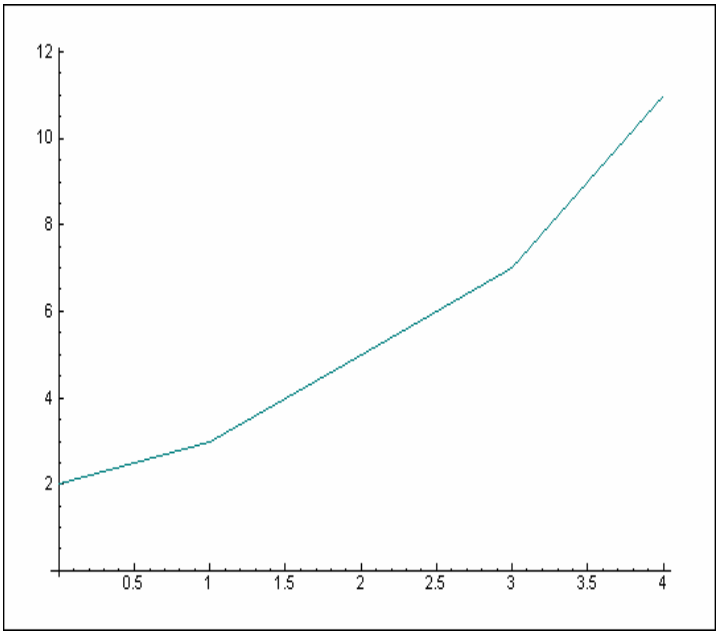
```
red=:<255 0 0
gdopen 'a'
red POLY SCALE < reg 6
gdshow''
gdopen 'b'
red POLY SCALE < 1 A. reg 6
gdshow''
gdopen 'c'
red POLY SCALE < /:~ reg 6
gdshow''

<@reg"0 i.6
```

D. Plotting Functions

This section illustrates the use of various facilities for plotting functions:

```
load 'plot'
plot x=:2 3 5 7 11
```

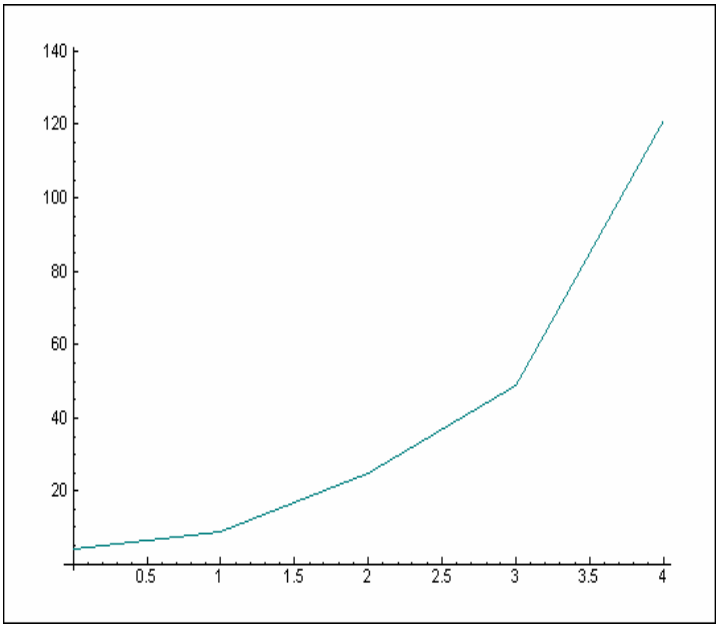


In this and the following plot, the horizontal axis is labeled with the default values from `0` to `4`. The next plot after that uses the form `x;*:x` to label this axis according to the argument `x`.

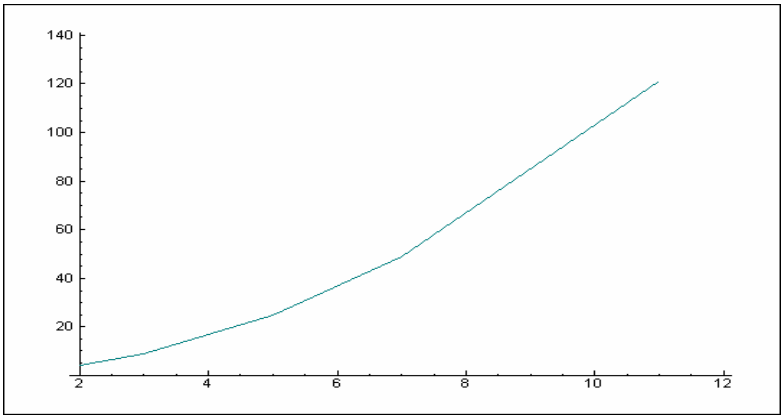
The alternative function **PLOT=:'stick,line'&plot** draws vertical "sticks" to each point as well as the "lines" between the points. Similarly, **BAR=:'stick'&plot** produces barcharts.

Enter the definitions of these functions, and experiment with them.

```
   plot *: x NB. Plot square function
```




```
   plot x;*: x NB. square Versus argument
```

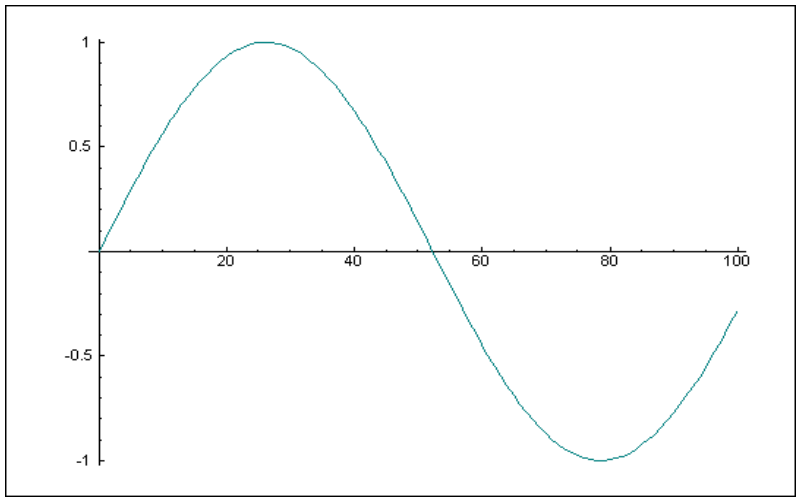


Entering **load 'graph'** also makes available a function called **steps** that produces a grid from one value to another in a specified number of steps. For example:

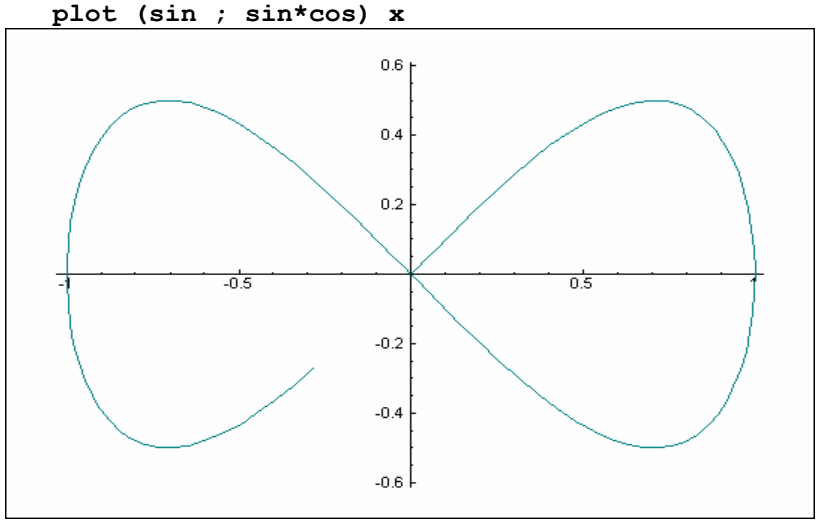
```
   steps 2 4 10
2 2.2 2.4 2.6 2.8 3 3.2 3.4 3.6 3.8 4
```

We will give it an alternative name as follows:

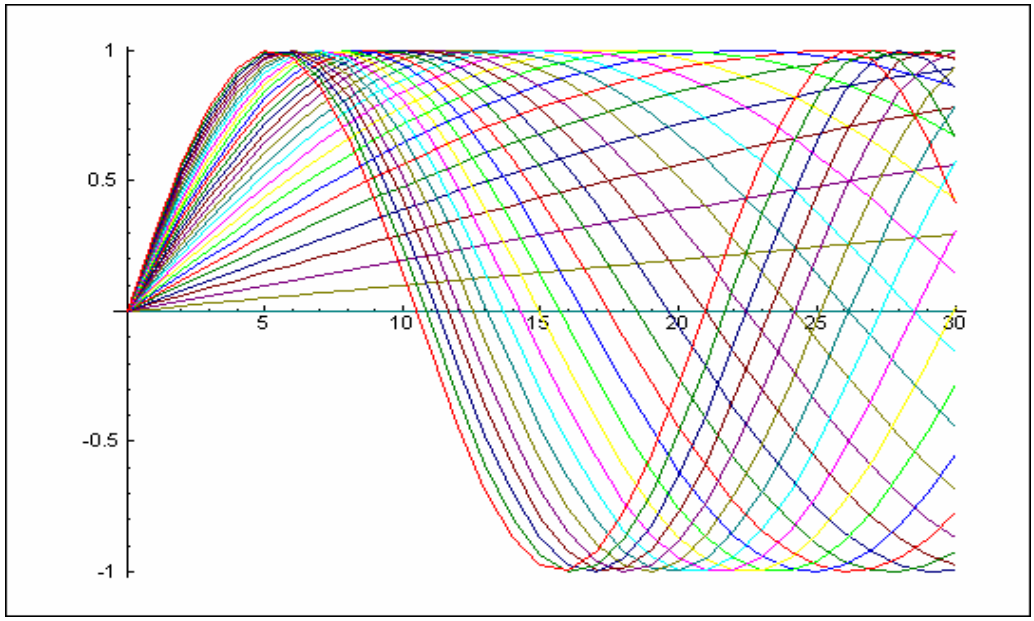
```
   grid=:steps
   grid 2 4 10  NB. 2 to 4 in 10 steps
2 2.2 2.4 2.6 2.8 3 3.2 3.4 3.6 3.8 4
   sin=:1&o.
   cos=:2&o.
   plot sin x=:grid 0 6 100
```



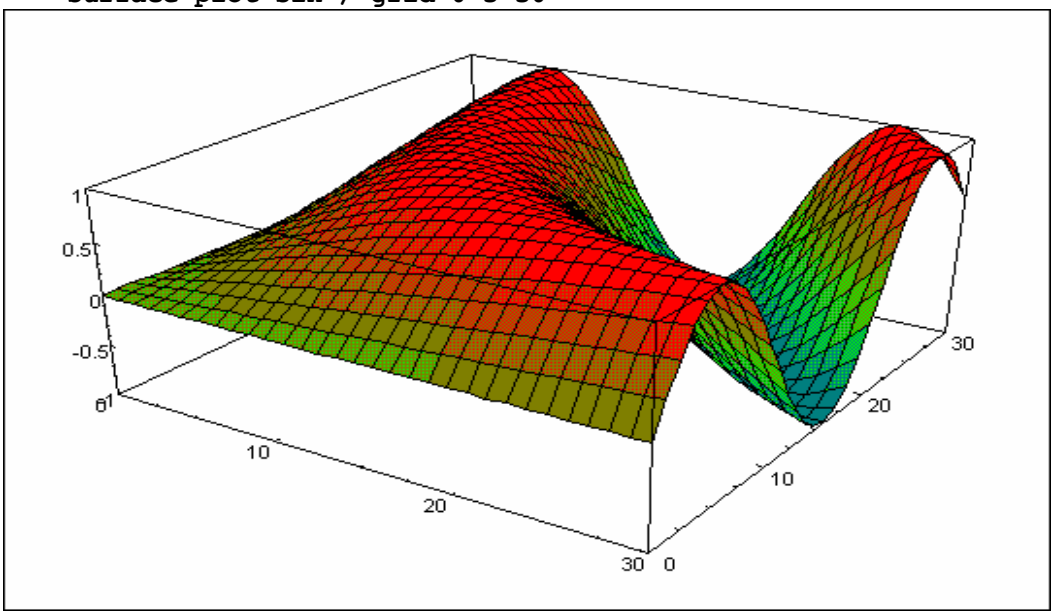

```
plot (sin ; sin*cos) x
```




```
plot sin */~ grid 0 3 30 NB. Multiple sines
```



```
'surface'plot sin*/~grid 0 3 30
```



Chapter 12

Linear Functions

That wholly consisted
of lines like these
C.S. Calverley

A. Distributivity

The properties of commutativity and associativity introduced in Chapters 3 and 9 concerned a single function; the important property of distributivity concerns a pair of functions. It is commonly treated as a relation between two dyadic functions, as illustrated below:

```
   'abc'=: 3 4 5                Assign the names a and b and c
   a,b,c
3 4 5

   ]d=:a*(b+c)
27

   ]e=:(a*b)+(a*c)
27
```

The general equivalence of the results `d` and `e` is expressed by saying that times *distributes* over addition. However, this distributivity might equally be expressed with the sum as the left argument of times as follows:

```
   ]f=:(b+c)*a
27
   ]g=:(b*a)+(c*a)
27
```

Times also distributes over subtraction, a fact that may be illustrated as follows:

```
   (a*(b-c));((a*b)-(a*c));((b-c)*a);((b*a)-(c*a))
+--+--+--+--+
|_3|_3|_3|_3|
+--+--+--+--+
```

Does division distribute over addition? It can be tested as follows:

```
   (a%(b+c));((a%b)+(a*c));((b+c)%a);((b%a)+(c%a))
+--------+-----+-+-+
|0.333333|15.75|3|3|
+--------+-----+-+-+
```

The result is conflicting; one pair agrees, and the other does not; a matter sometimes resolved by saying that division distributes *to the left*, but not to the right. It is simpler and clearer to note that the *monadic* function **a&%** does not distribute over addition, but that the function **%&a** does. We will hereafter speak only of the distributivity of monadic functions. For example, **+:** (double) and **-:** (halve) both distribute over addition.

Exercises

1. Does **%&a** distribute over subtraction? Test your assertion.
2. Repeat the experiments of this section using conformable (that is, equal in number of items) lists **a**, **b**, and **c**.
3. Repeat the experiments of this section using conformable tables **A**, **B**, and **C**.
4. Experiment with the dyadic cases of the functions **f@g** and **g&f** for various values of the proverbs **f** and **g** (such as **f=:%** and **g=:-**), and state clearly the effects of the conjunctions **@** and **&**

   [**b f@g c** is equivalent to **f b g c**, and
   b f&g c is equivalent to **(g b)f(g c)**]

5. Comment on the assertion that the equivalence of **f@g** and **g&f** is a test of the distributivity of **f** over **g**
6. Experiment with the conjunction **dtest=: 12 : 'x.@y. -: y.&x.'** in testing for distributivity. Include **+dtest-** and **b %&3 dtest + c** and **b 3&% dtest + c**

B. Linearity

A function that distributes over addition is said to be *linear*. Linear functions prove to be important in almost every branch of applied math.

The functions **L1=:*&2** and **L2=:%&2** and **L3=:|."1** are each linear. Thus:

```
   a=:3 4,9 4,:9 7 [ b=:3 4,9 4,:6 8

   a;b
+---+---+
|3 4|3 4|
|9 4|9 4|
|9 7|6 8|
+---+---+

   L1 (a+b)
12 16
36 16
```

```
30 30

   (L1 a)+(L1 b)
12 16
36 16
30 30
```

Such matters may be expressed more clearly and compactly as follows:

```
   a (L1@+ ; +&L1) b
+-----+-----+
|12 16|12 16|
|36 16|36 16|
|30 30|30 30|
+-----+-----+

   a (L1@+ -: +&L1) b
1
```

C. Linear Vector Functions

A function of rank 1 applies to each vector in its argument, and may be called a *vector function*. We will use the term in a more restrictive sense: the result must be the same shape as the argument. Thus `L3=:|."1` defined in the preceding section is a linear vector function:

```
   d=: 4 2 1 [ e=: 2 3 5
   L3 d+e
6 5 6
   (L3 d)+(L3 e)
6 5 6
```

If `f=:+/@:*"1`, then the function `w&f` is a *weighted sum*, with weights specified by the vector `w`. Moreover, it is linear. For example:

```
   w=:2 0 3
   w&f d
11
   w*d
8 0 3

   +/w*d
11

   (w&f d+e),:(w&f d)+(w&f e)
30
30
```

Although `w&f` is linear, it is not a linear vector function according to our strict definition. Such a linear vector function may be defined as follows:

```
   x=:5 1 2
```

```
   y=:7 2 0
   g=: w&f,x&f,y&f
   g d
11 24 32

   t=:w,x,:y
   t
2 0 3
5 1 2
7 2 0

   h=: t&f
   h d
11 24 32
```

In general, if **t** is an n-by-n table, then **t&f** is a linear vector function on any vector of n elements.

Exercises

7. Use the arguments **x** and **y** to test the assertions that each of the following functions is linear:

```
   x=:2 7 1 8
   y=:3 1 4 2
   L4=:+/\
   L5=:L4*L4
   L6=:L4^:_1
```

   [**L5** is not linear. **L6** illustrates the fact that the inverse of a linear function is linear. **L4** gives subtotals, and **L6** gives differences: try **L4 L6 x** and **L6 L4 x** to test the assertion that they are inverse functions.]

D. Inner Product

Applied to the sum (**+/**) and product (*****), the *dot* conjunction produces the *matrix product* function that is (for the arguments used in the preceding section) equivalent to the function `f` defined there:

```
   mp=:+/ . *        The space before the dot is essential
   w mp d            Using w and d and t from the preceding section
11

   t &mp d
11 24 32
```

For any *square* matrix **m** (that is, **=/$m**), the function **mp&m** is a linear vector function. For example:

```
   m=:5-~?.4 4$10
   L=:m&mp"1
   x=:2 7 1 8
```

```
   y=:3 1 4 2

   m;(L x);(L y);(L x+y);(L x)+(L y)
+----------+-----------+-------------+-------------+-------------+
|_4  2 _1 0|           |             |             |             |
|_3 _5  1 1|5 _32 18 _37|_14 _8 16 _18|_9 _40 34 _55|_9 _40 34 _55|
| 4 _2  0 3|           |             |             |             |
|_5 _5  0 1|           |             |             |             |
+----------+-----------+-------------+-------------+-------------+
```

Conversely, for any linear function, a matrix `m` that represents it (in the function `L=:m&mp"1`) can be obtained by simply transposing the table produced by applying the function to the appropriate *identity* matrix. For example:

```
   (];L;|:@:L;m"_) I=:=i.4
+-------+-----------+----------+----------+
|1 0 0 0|_4 _3  4 _5|_4  2 _1 0|_4  2 _1 0|
|0 1 0 0| 2 _5 _2 _5|_3 _5  1 1|_3 _5  1 1|
|0 0 1 0|_1  1  0  0| 4 _2  0 3| 4 _2  0 3|
|0 0 0 1| 0  1  3  1|_5 _5  0 1|_5 _5  0 1|
+-------+-----------+----------+----------+

   (|:L I)&mp x
5 _32 18 _37

   L x
5 _32 18 _37
```

Exercises

8. Using the result of `L4 x` from Exercise 7, try to determine by hand the value of the matrix `m` such that `m&mp x` gives the same result. Compare your result with
   `|:@:L4 I=:=i.4`.
9. Compare the results of the function `m&mp` derived in Exercise 8 with the result of `L4` when applied to the argument `z=:2 7 1 8 2 8`.
10. Repeat Exercise 8 for the function `L6`.
11. Repeat Exercise 8 for the function `L3=:|."1`.

The error produced in Exercise 9 illustrates the fact that the domain of the matrix product representation of a linear function is restricted to arguments of a specific number of items, even though the linear function from which it is derived has a wider domain.

E. Why The Name “Linear”?

Why is a function that distributes over addition called *linear*? We will attempt to answer this by applying an arbitrary linear vector function to geometric figures, beginning with the right-angled and isosceles triangles of Chapter 11:

```
   is=:3 4,9 4,:6 8
   rt=:3 4,9 4,:9 7
   ]m=:?.2 2$10
1 7
4 5
```

```
   mp=:+/ . *
   L=:m&mp"1
   rt;L rt
+---+-----+
|3 4|31 32|
|9 4|37 56|
|9 7|58 71|
+---+-----+

   is;L is
+---+-----+
|3 4|31 32|
|9 4|37 56|
|6 8|62 64|
+---+-----+
```

We may plot these resulting triangles (by hand or by the methods of Chapter 11) to try to assess the effects of the linear function. Is the right-angle of `rt` retained? Do the two equal sides of `is` remain equal? Is the order of the vertices reversed? We may also apply the function `AREA` of Chapter 11 to compare the areas:

```
   AREA=:det@(,"1&0.5)
     det=:-/ . *
   (AREA L rt)%(AREA rt)
_23

   (AREA L is)%(AREA is)
_23
```

The areas of the two triangles appear to be multiplied by the same factor. In fact, the *area transformation* produced by a function `m&mp` is the determinant of `m` :

```
   det=:-/ . *
   det m
_23
```

We now consider three points on a line, that is, a *degenerate* triangle having zero area:

```
   a=:3 4
   b=:5 13
   ]deg=: a,b,:(a%4)+(3*b%4)
  3     4
  5    13
4.5 10.75

   AREA deg
0

   AREA L deg
0
```

This result suggests (correctly) that a linear function transforms a line into a line, a fact that suggests the use of the term *linear* for it.

A point in three-dimensional space can be represented by a three-element vector such as `p=:3 1 5`. A linear function on such a point must, of course, be represented by a 3-by-3 matrix `m`. Moreover, a tetrahedron may be represented by a 4-by-3 table, and the function **AREA** may be modified to give its volume as follows:

```
VOL=:det@(,"1&1r6)
```

Exercises

12. Use a tetrahedron (such as `tet=:0 0 0,0 0 1,0 1 0,:1 0 0`) whose volume is easily computed to test the behaviour of the function **VOL**.
13. Use a permutation of the vertices of `tet` to show that **VOL** gives the signed volume of its argument much as **AREA** does. State the condition for a non-negative volume.

    [Try `VOL 1 A. tet`. The volume is non-negative if the vertices of the "base" triangle are in counter-clockwise order when viewed from the leading vertex]
14. Use expressions analogous to those used for the area of a triangle to investigate the volume transformation effected by a linear function on a tetrahedron.
15. Define a degenerate tetrahedron (in which the four points are co-planar) to illustrate the fact that a linear function on it yields a co-planar result.

Chapter 13

Representations of Functions

No computation
without representation
Adin Falkoff

A. Introduction

A family of monadic functions is commonly represented by a single dyadic function, a particular member of the family being obtained by bonding a parameter. As an example, consider the *permutation* or *anagram* function introduced in Chapter 2:

```
   a=:'ABCDE'
   2 A. a
ABDCE

   f=:2&A.
   f a
ABDCE
```

A family may also be represented in several ways, using different dyadic functions. For example:

```
   0 1 4 3{a
ABED
```

The *indexing* or *from* function

```
   p=:0 1 3 2 4
   p{a
ABDCE
```

A *permutation vector* (a permutation of `i.5`)

```
   p&{ a
ABDCE
```

A monadic permutation function

```
   ]b=:0;1;3 2;4
+-+-+---+-+
|0|1|3 2|4|
+-+-+---+-+

   b C. a
ABDCE
```

The cycle function `C.`

```
   b&C. a                    A monadic permutation function
ABDCE
```

Since different representations have different uses, it is important to provide transformations from one to the other. The monadic cases of `A.` and `C.` provide such transformations:

```
   A. p
2

   (A. p) A. a
ABDCE

   ]b=: C. p
+-+-+---+-+
|0|1|3 2|4|
+-+-+---+-+

   C. b
0 1 3 2 4
```

The behaviour of these various representations of permutations can be studied by using random permutations generated by the function `?.~`. For example:

```
   ]q=:?.~9
7 1 3 2 6 4 0 5 8

   A. q
288918

   (A.q)A. i.9
7 1 3 2 6 4 0 5 8

   C. q
+-+---+---------+-+
|1|3 2|7 5 4 6 0|8|
+-+---+---------+-+
```

Exercises

1. Generate a table of all permutations of order 4.
   [`(i.!4)A. i.4`]
2. Use the example of `q=:?.~9` and `C.q` to illustrate the scheme used in the cycle representation of permutations.

   [The third box of `C.q` signifies that item 5 moves to position 7, item 4 to position 5, item 6 to 4, item 0 to 6, and item 7 to 0. Moreover, item 8 moves to 8 (and therefore remains fixed). Use the help menu for discussion of permutations in the introduction to the dictionary, the vocabulary, and the phrase book.]
3. Is a permutation a linear function? If it is, produce the matrix `m` that represents it in the expression `m&(mp=:+/ . *)` .

```
[m=:q=/i.#q]
```

Chapter 14

Polynomials

A. Coefficients Representation

A function that is a multiple of a non-negative integral power of its argument is called a *monomial*. In MN it is written in the form $3x^2$, yielding the value 12 if the argument x has the value 2.

A sum of monomials is called a *polynomial*, and is written in MN in the form $2x^0+4x^1+3x^2+x^3$, having the value 30 if x is 2.

A direct translation to J would read as `(2*x^0)+(4*x^1)+(3*x^2)+x^3`. The numerous parentheses required suggest a reason for the precedence rules adopted in MN (power before multiplication before addition); they are precisely the rules that permit the polynomial to be expressed without parentheses.

Exercises

1. Write a parenthesis-free J expression for the foregoing polynomial, then assign the value `2` to `x` and enter the expression to test its validity.

2. Use the results of Exercise 1 to define a function `py` so that `2 4 3 1 py x` yields the value of the polynomial for any single argument `x`.

   [`py=:+/@([ * ] ^ i.@#@[)`]

3. Use `py` to define a function `poly` so that it applies to each element of a list `x`, and test it by using it with the arguments `2 4 3 1` and `i.8`.

   [`poly=:py"1 0`]

4. Comment on the function `2 4 3 1&poly`.

   [The dyadic function `poly` represents a family of polynomials, `2 4 3 1&poly` is a specific member of this family. The elements of the list `2 4 3 1` are called coefficients, and `poly` is said to be a coefficients representation of polynomials.]

5. The dyadic case of the primitive function `p.` is a coefficients representation of polynomials. Experiment with the expression `c p. x` for various values of `c` and `x`.
6. Experiment with `tc p./ x`, where `tc` is a table of coefficients and `x` is a list.

B. Roots Representation

The product `*/x-r` is said to be a polynomial expressed in terms of the list of *roots* `r`. It is called a polynomial because any such function can also be expressed in a coefficients representation. Thus:

```
   x=:4
   r=:2 3 5
   x-r
2 1 _1

   */x-r
_2

   _30 31 _10 1 p. x
_2
```

The monadic case of `p.` applied to the boxed roots yields the coefficients of the other representation:

```
   p. <r
_30 31 _10 1
```

Exercises

7. Define a "polynomial in terms of roots" function `pir` such that `r&pir x` evaluates a polynomial with roots `r` for the argument `x`.
8. Why are the elements of the list `r` in the function `r&pir` called roots?

   [Each of the elements is a zero or root of the function in the sense that it yields a zero result. For example, enter `pir=: */@(]-[)"1 0` and `r&pir r`]
9. Every function of the form `r&pir` can be represented in the form `c&p.`. Is the converse true?

   Try to define a list `s` such that `s&pir` is equivalent to `d&p.`, where `d=:2*p.<r`. Then look at the result of `p. d` and of `(p. d) p. x=:i.8`]
10. Discuss the result of `p. d`.

    [The dyadic function `p.` covers both the coefficients and roots representations. If the left argument is open (not boxed), it is treated as a list of coefficients. If it is boxed (and contains two items), the last item is the boxed list of roots, and the first is the boxed "multiplier". If it contains a single item `b`, it is equivalent to `1;b` (that is, a multiplier of `1`).]

C. Versatility

The polynomial is a most important function in math. This importance stems from its versatility, which in turn stems from a few simple properties.

The discussion of these properties leads to a number of topics not yet discussed, such as complex numbers, derivatives, power series, and transcendental functions (including the exponential (`^`), sine (`1&o.`), and cosine (`2&o.`)). Even if you are unfamiliar with such matters, you will probably find it fruitful and interesting to use this section as an introduction to them, always remembering the injunction of Chapter 1: do not spend too much time on matters that may be, at the moment, beyond your powers.

In presenting the properties of polynomials we will use the following in examples:

```
   c=:1 3 3 1
   d=:2 1 0 4
   s=:c+d
   p=:+//.c*/d
   c;d;s;p
+-------+-------+-------+---------------+
|1 3 3 1|2 1 0 4|3 4 3 5|2 7 9 9 13 12 4|
+-------+-------+-------+---------------+
```

- The sum (or difference) of two polynomials is itself a polynomial. For example, the polynomial `f=:c&p.+d&p.` is equivalent to the polynomial `g=:(c+d)&p.`.
- The product of polynomials is a polynomial: `c&p.*d&p.` equals `p&p.`
- Polynomials can be used to approximate a wide variety of important functions. A *power series* is a polynomial whose coefficients are each expressible as a function of its index. For example, the reciprocal factorial function `expc=:%@!` specifies the power series approximation to the exponential funtion. Thus:

```
      expc=:%@!
    ]e8=:expc i.8
1 1 0.5 0.1666667 0.04166667 0.008333333

   e8&p. i.4
1 2.71667 7.26667 18.4

   ^i.4
1 2.71828 7.38906 20.0855
```

- The *derivative* (that is, the rate of change or slope of the tangent to the graph) of a polynomial is itself a polynomial. For example, the derivative of `c&p.` is `(1 }. c * i.#c)&p.`
- The *integral* (or anti-derivative) of a polynomial is itself a polynomial. For example the integral of `c&p.` is `(0 , c % 1+i.#c)&p.`.
- The composition `(c&p.)@(d&p.)` is a polynomial.

Exercises

11. Experiment with the foregoing examples.
12. Define and use `plus=:+` and `times=:+//.@(*/)` and `der=:1: }. ] * i.@#` and `int=:0: , ] % 1: + i.@#` . Comment on their behaviour.

    [`der@int` is an identity function. The function `plus` fails for arguments that do not have the same number of items. Try the function `plus=:+/@,:` and examine how the laminate function pads a shorter argument with (non-significant) trailing zeros]
13. Explain the reason for the diagonal sums (produced by `/.`) used in the function `times`.

    [See the multiplication of decimal numbers in Section C of Chapter 7]

The *Taylor series* adverb `t.` produces a function that gives the coefficients of a power series. For example:

```
   c&p. t. i.8
1 3 3 1 0 0 0 0

   (c&p.*d&p.) t. i.8
2 7 9 9 13 12 4 0

   ^ t. i. 7
1 1 0.5 0.1666667 0.04166667 0.008333333 0.001388889
   sin=:1&o.
   cos=:2&o.
   ]sc=:sin t. i.8
0 1 0 _0.1666667 0 0.008333333 0 _0.0001984127

   ]cc=:cos t. i.8
1 0 _0.5 0 0.04166667 0 _0.001388889 0
```

The power series for an ordinary polynomial (that is, one with a finite list of coefficients) ends with (non-significant) zeros, but the series for a transcendental function continues with non-zero terms. However, the coefficients for the exponential, sine, and cosine diminish rapidly in magnitude. This rapid decline accounts for the utility of a small number of terms in approximating functions.

Exercises

14. Predict and confirm the results of `((cos*cos)+(sin*sin))t. i.8`
15. Repeat Ex 14 for `((cc times cc)plus(sc times sc))t. i.8`
16. Repeat Ex 14 for `(^t.i.8)times(^@-t.i.8)`
17. The function `h=:(1 2 3&p.)@(4 3&p.)` is a polynomial. Determine its coefficients by hand, and confirm the result by entering `h t. i.8`.
18. Read Section 9D (Expansion) of Book 2.

If **f** and **g** are polynomials, then **(f*g)%g** is equivalent to **f**. On the other hand, division for an arbitrary pair (such as **f%g**) may be not a polynomial, but a power series. For example:

```
   f=:1&p.
   g=:1 _1 _1&p.
   (f%g) t. i. 8
1 1 2 3 5 8 13 21
```

The foregoing Taylor series may be surprising: it is the *Fibonacci* series, in which each item is the sum of the two preceding it. This matter is discussed in *Concrete Mathematics* [3], and in *Concrete Math Companion* [2].

D. Parity

A function **E** is said to be even if **E** is equivalent to **E@-**, that is, **E x** equals **E-x** for any argument **x**. Graphically this implies that the graph of an even function is reflected in the vertical axis.
A function **o** is *odd* if **o** is equivalent to **-o@-**, that is, **o x** equals **-o-x** for any **x**. Consequently, the graph of an odd function is reflected in the origin.

Exercises

19. What is the parity (odd or even) of each of the functions sine and cosine?
20. Enter **sin t. i.8** and **cos t. i.8** and comment on the power series of odd and even functions.

[The coefficients of all odd powers of an even function are zero, and conversely.]

21. What are the parities of the products of an even function with an even; an even function with an odd; an odd with an odd? Test your assertions.
22. What is the parity of the exponential function?

The exponential is an example of a function that is neither odd nor even. However, any function can be expressed as the sum of two functions, an odd part and an even part. For example:

```
   opex=:2: %~ ^ - ^@-

   epex=:2: %~ ^ + ^@-

   (^,opex,epex,:opex+epex) i.8
1 2.71828 7.38906 20.0855 54.5982 148.413 403.429 1096.63
0  1.1752 3.62686 10.0179 27.2899 74.2032 201.713 548.316
1 1.54308  3.7622 10.0677 27.3082 74.2099 201.716 548.317
1 2.71828 7.38906 20.0855 54.5982 148.413 403.429 1096.63
```

The odd and even parts of a function may be functions of interest in their own right. In the present case, **opex** and **epex** are the hyperbolic sine and hyperbolic cosine (often abbreviated as sinh and cosh); denoted in J as illustrated below:

```
   (5&o.,:6&o.)i.8
0  1.1752 3.62686 10.0179 27.2899 74.2032 201.713 548.316
1 1.54308  3.7622 10.0677 27.3082 74.2099 201.716 548.317
```

The adverbs `o=: .:-` and `E=: ..-` produce the odd and even parts of functions to which they are applied. For example, `^o` is equivalent to `opex` and `^E` is equivalent to `epex`.

Exercises

23. Compare the coefficients `^t.i.8` and `opex t.i.8` and `epex t.i.8` .

24. Comment on the functions `cos E` and `cos o` and `sin E` and `sin o` .

25. The function `j.` multiplies its argument by `0j1`, the "imaginary" square root of negative `_1`. Comment on the even function `^@j.E` .
    [`^@j.E` is the cosine. Try entering `^@j. t. i.8` and `^@j.E t. i.8`]

E. Linearity

Since `(c+d)p. x` equals `(c p. x)+(d p. x)`, it appears that a polynomial is in some sense a linear function of its coefficients. We will now consider a series of examples to clarify this vague statement, producing the matrix that represents the linear function, and a simple expression for it as a power table:

```
   mp=:+/ . *
   c=:1 3 3 1 [ d=:2 1 0 4 [ x=:1 2 3 4
   (c p. x);(c&p. x);(p.&x c)
+-----------+-----------+-----------+
|8 27 64 125|8 27 64 125|8 27 64 125|
+-----------+-----------+-----------+

   I=:=i.4
   I
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

   ]m=:|:p.&x"1 I
p.&x
1 1  1  1
1 2  4  8
1 3  9 27
1 4 16 64

   m mp c
8 27 64 125

   m&mp c
8 27 64 125
```

The matrix that represents the linear function

The matrix `m` that represents the desired linear function of the coefficients looks like a power table, and may be so expressed in terms of the argument `x` and its indices as follows:

```
   ]e=:i.#x
0 1 2 3
```

```
   x ^/ e
1 1  1  1
1 2  4  8
1 3  9 27
1 4 16 64
```

The table **m** is called the (Complete) Vandermonde matrix of **x**. A Vandermonde function may be defined and used as follows:

```
   V=:] ^/ i.@[          The Vandermonde function
   (#x) V x              Vandermonde matrix for x
1 1  1  1
1 2  4  8
1 3  9 27
1 4 16 64

   ]y=:((#x) V x)&mp c   Linear function in terms of Vandermonde
8 27 64 125

   f=:c&p.
   f x
8 27 64 125

   CV=:# V ]             Complete Vandermonde function

   ]y=:(CV x)&mp c
8 27 64 125
```

The complete Vandermonde matrix is square and invertible. Its inverse provides the inverse linear function, which may be used to determine the coefficients of a polynomial that represents the function as illustrated below:

```
   %. CV x
   4  _6    4         _1
  _4.33333 9.5   _7   1.83333

       1.5  _4  3.5         _1
_0.1666667 0.5 _0.5 0.1666667

   (%.CV x)&mp y         The inverse linear function applied to y
1 3 3 1

   (%.CV x)&mp f x       Using the fact that y is f x
1 3 3 1

   f                     Show the definition of f
1 3 3 1&p.
```

Exercises

26. Use the foregoing discussion as a model for experimenting with Vandermonde matrices for various values of the arguments **x** and **c**, and comment on the results.

    [The linear function **(CV x)&mp** applies only to arguments

that have the same number of items as does **x**.]

27. Use **x=:10%~i.10** and **y=:(sin=:1&o.) x** to obtain coefficients **c** such that **c&p.** agrees with **sin** for the arguments **x**. Use expressions of the form **(c&p. ; sin) 5 2$x** to show the comparison clearly.

28. Test the use of **c&p.** to approximate **sin** by evaluating **(c&p.-sin)z** for other arguments such as **z=:0.65 0.8** and **z=:i.5 2** and comment on the results.

    [The approximation is good in the range covered by **x** (0-0.9), but may be very bad for arguments outside this range.]

F. Polynomial Approximations

Sections C and E have presented two methods of approximating a function **f** by a polynomial. The first used the Taylor series **f t. i. n**, and the second the complete Vandermonde matrix **CV x** to fit the function exactly at the points **f x**. We will first compare their results for the example treated in Exercise 27:

```
   '`sin mp CV'=: (1&o.)`(mp=:+/ . *)`(# (V=:] ^/ i.@[) ])
   x=:10%~i.10
   tc=:sin t. i. # x
   vc=:(%.CV x) mp sin x

   ((sin-tc&p.);(sin-vc&p.)) 5 2$x
+------------------------+------------------------+
|           0            0|_2.18587e_13  1.5066e_12|
|_5.27356e_16 _4.43534e_14|_4.92267e_12 1.02958e_11|
|_1.04966e_12 _1.22129e_11|_1.40716e_11 1.53755e_11|
|_9.06789e_11  _4.9381e_10|_4.78062e_12 1.69003e_11|
| _2.14316e_9  _7.82095e_9| 3.04181e_11  6.7725e_11|
+------------------------+------------------------+

   ((sin-tc&p.);(sin-vc&p.)) 5 2$x+0.1
+------------------------+------------------------+
|           0 _5.27356e_16| 1.5066e_12 _4.92267e_12|
|_4.43534e_14 _1.04966e_12|1.02958e_11 _1.40716e_11|
|_1.22129e_11 _9.06789e_11|1.53755e_11 _4.78062e_12|
| _4.9381e_10  _2.14316e_9|1.69003e_11  3.04181e_11|
| _7.82095e_9  _2.48923e_8| 6.7725e_11   1.05946e_8|
+------------------------+------------------------+
   ((sin-tc&p.);(sin-vc&p.)) 5 2$-x
+-----------------------+-----------------------+
|          0           0|_2.18587e_13  _1.07342e_8|
|5.27356e_16 4.43534e_14| _9.83991e_8   _5.0716e_7|
|1.04966e_12 1.22129e_11| _1.92828e_6  _6.01583e_6|
|9.06789e_11  4.9381e_10| _1.62857e_5  _3.95772e_5|
| 2.14316e_9  7.82095e_9| _8.83031e_5 _0.000183771|
+-----------------------+-----------------------+
```

The first panel above shows that **vc** provides the better approximation at the very points on which it was determined; the second panel shows that this better performance persists for other points in the range spanned by them; and the third shows that the Taylor series generally performs better at points (that is, **-x**) outside the range.

Exercises

29. Will the use of a larger number of terms in a polynomial approximation improve its fidelity? Experiment to test the matter.

    [Not necessarily. Although the higher-order elements of the coefficients `f t. i.n` may decrease rapidly, the power of the argument by which each is multiplied in the polynomial evaluation may rapidly increase. The resulting product produced (to limited precision) may introduce large "round-off" errors.]

We will now develop a polynomial of lower degree that provides a "least-squares best fit" to the values `f x`. With a left argument less than `#x` the Vandermonde function `V` produces non-square power tables as illustrated below:

```
   x=:1 2 3 4
   (1&V;2&V;3&V;4&V;CV) x
+-+---+------+---------+---------+
|1|1 1|1 1  1|1 1  1  1|1 1  1  1|
|1|1 2|1 2  4|1 2  4  8|1 2  4  8|
|1|1 3|1 3  9|1 3  9 27|1 3  9 27|
|1|1 4|1 4 16|1 4 16 64|1 4 16 64|
+-+---+------+---------+---------+
```

Although these matrices are not square, they may be used with the generalized inverse function denoted by `%.` as illustrated below:

```
   %.3 V x
 2.25 _0.75 _1.25  0.75
_1.55  1.15  1.35 _0.95
 0.25 _0.25 _0.25  0.25

   %.2 V x
   1  0.5   0 _0.5
_0.3 _0.1 0.1  0.3

   f=:(c=:1 3 3 1)&p.
   ]vc3=:(%. 3 V x) mp f x
11.5 _13.7 10.5

   vc3 p. x
8.3 26.1 64.9 124.7

   c p. x
8 27 64 125
```

The matrix product `(%. 3 V x) mp f x` used above can be written more simply as a "matrix divide", by using the dyadic case of the function `%.` . Thus:

```
   (f x)%.(3 V x)
11.5 _13.7 10.5
```

Finally we define a conjunction `FIT` such that `n FIT f x` gives an `n`-element list of coefficients that fits the function `f` at the points `x`. Thus:

```
   FIT=:2 : 'y. %. (x. & V)'
   3 FIT ^
^ %. 3&V

   c=:3 FIT ^ y=:0.1*i.7
   c p. y
1.00133 1.10388 1.22004 1.3498 1.49317 1.65015 1.82073
   ^y
1 1.10517 1.2214 1.34986 1.49182 1.64872 1.82212

   f
1 3 3 1&p.

   d=:3 FIT f x
   d p. x
8.3 26.1 64.9 124.7

   f x
8 27 64 125
```

Exercises

30. Experiment with the conjunction **FIT** for various values of its parameters. Include the example used at the beginning of this section, and compare the fit provided by the coefficients **tc** with that provided by the five-element list **tc5=:5 FIT sin x**.

Chapter 15

Arithmetic

A. Introduction

As remarked in Chapter 1, arithmetic is that branch of mathematics that deals with whole numbers. As treated in Book 2, it includes topics such as permutations, polynomials, and logic. These are usually considered to be advanced topics, to be treated only after the introduction of fractions, irrational numbers, and even complex numbers. What are the potential advantages of extending the treatment of arithmetic in this manner?

- It may serve to defer the treatment of fractions until the student has matured through experience gained in many interesting uses of whole numbers. How many cooks fear the use of fractions involved in dividing a recipe? Is 2/3 really a number since it cannot be written in decimal, although 3/4 can? And how many question the point of complex numbers whose mechanics are often elaborated long before any of their interesting uses can be shown?
- Although polynomials may be of little practical use when limited to integer arguments, notions such as the product of coefficients (`+//.c*/d`) remain meaningful and interesting. Indeed they provide useful insights into the products of multi-digit numbers, as shown in Chapters 7 and 14.

B. Insidious Inverses

The familiar *counting* numbers may be defined as follows: there is a first (denoted by `1`), and a *successor* function (denoted by `>:`). Thus:

```
   >:1
2

   >:2
3

   >:>:>:1
4
```

An inverse *predecessor* function (denoted by `<:`) undoes the work of the successor. Thus:

```
   <:4
3

   <:>:3
3
```

However, `<:` is not a *proper* inverse, because its application to the *first* counting number cannot yield a counting number. Thus:

```
   <:1
0

   <:0
_1

   <:_1
_2
```

In other words, the introduction of a seemingly-innocent inverse has broadened the class of counting numbers to define the class of *integers*, which includes zero and negative numbers. The introduction of the further classes of rationals, irrationals, and complex numbers can be viewed in a similar light.

Exercises

1. Illustrate the fact that the successor and predecessor are proper inverses on the domain of integers. Include examples of the powers `<:^:n` and `>:^:n` for both positive and negative values of `n`.
2. Same and illustrate the use of a function that has a proper inverse on some domain.

   [On the domain of permutation vectors (permutations of the integers `i.n`), the grade (`/:`) is its own proper inverse.]
3. Experiment with some of the inverse pairs listed in the definition of the power conjunction (`^:`) in the J dictionary [5].
4. Read the discussion in the first three pages of Book 2.
5. Study Section 2 I (Identity Elements and Infinities) of Book 2.

C. Rational Numbers

The multiplication of two integers yields an integer. Moreover, division is inverse to multiplication in the sense illustrated below:

```
   _2*8
_16
```

```
   (_2*8)%8
_2
```

More precisely, if `i` is an integer, then the functions `*&i` and `%&i` are inverse:

```
   i=:8
   *&i _2
_16

   %&i *&i _2
_2
```

Again, `%&i` is not a proper inverse because it may lead out of the class of integers, producing the class of *rationals*. For example:

```
   %&i _2
_0.25
```

Exercises

6. Illustrate the fact that the rationals are closed under multiplication and division.

D. Irrational Numbers

The square function is closed on the rationals, and the square root (`%:`) provides an inverse. For example:

```
   ]r=:3%5
0.6

   *:r
0.36

   %: *: r
0.6
```

Again `%:` is not a proper inverse, because there is no rational whose square is `2`, and the result is to introduce a further class of *irrationals*. Because there is at least one rational between any pair of distinct rationals (their average), it might seem impossible that there could be any numbers that are *not* rational. However, the school of Pythagoras produced a rather straightforward argument to show that the square root of `2` (the length of the hypotenuse of a right-triangle with sides of unit length) is not a rational.

E. Complex Numbers

Because there is no rational whose square is negative, the square root applied to a negative argument leads to the further class of *complex* numbers. Thus:

```
   %:_1
0j1

   ]a=:i.6
0 1 2 3 4 5

   %:a
0 1 1.41421 1.73205 2 2.23607

   %:-a
0 0j1 0j1.41421 0j1.73205 0j2 0j2.23607
```

Taken together with the rationals, these *imaginary* square roots of negative numbers form the class of *complex* numbers, closed under square root as well as under addition, subtraction, multiplication, and division.

Exercises

7. Read Section 9F (Real and Complex Numbers) of Book 2.
8. Read Chapter 7 (Permutations) of Book 2.
9. Read Chapter 8 (Classification and Sets) of Book 2.

Chapter 16

Complex Numbers

A. Introduction

The following tables illustrate some of the consequences of adding the *imaginary* square root of minus one to the number system:

Bordered table adverb adapted from Ch. 3

```
   T=:1 : '[by]over x./' ~
     by=:[:":' '&;@,.@[,.]
     over=:({.;}.)@":@,

   ]i=:%:_1
0j1
   ]c=:(i.4),i*i.4
0 1 2 3 0 0j1 0j2 0j3
```

Addition table

```
   +T c
+---+-------------------------------+
|   |  0   1   2   3   0 0j1 0j2 0j3|
+---+-------------------------------+
|  0|  0   1   2   3   0 0j1 0j2 0j3|
|  1|  1   2   3   4   1 1j1 1j2 1j3|
|  2|  2   3   4   5   2 2j1 2j2 2j3|
|  3|  3   4   5   6   3 3j1 3j2 3j3|
|  0|  0   1   2   3   0 0j1 0j2 0j3|
|0j1|0j1 1j1 2j1 3j1 0j1 0j2 0j3 0j4|
|0j2|0j2 1j2 2j2 3j2 0j2 0j3 0j4 0j5|
|0j3|0j3 1j3 2j3 3j3 0j3 0j4 0j5 0j6|
+---+-------------------------------+
```

Multiplication table

```
   *T c
+---+---------------------------+
|   |0   1   2   3 0 0j1 0j2 0j3|
+---+---------------------------+
|  0|0   0   0   0 0   0   0   0|
|  1|0   1   2   3 0 0j1 0j2 0j3|
|  2|0   2   4   6 0 0j2 0j4 0j6|
|  3|0   3   6   9 0 0j3 0j6 0j9|
|  0|0   0   0   0 0   0   0   0|
|0j1|0 0j1 0j2 0j3 0  _1  _2  _3|
|0j2|0 0j2 0j4 0j6 0  _2  _4  _6|
|0j3|0 0j3 0j6 0j9 0  _3  _6  _9|
+---+---------------------------+
```

```
   %:T 1 2 3 0j1 0j2 0j3          Roots
+---+-----------------------------------------------------------------------------------------+
|   |1                 2                 3              0j1              0j2               0j3|
+---+-----------------------------------------------------------------------------------------+
|  1|1                 2                 3              0j1              0j2               0j3|
|  2|1           1.41421           1.73205 0.707107j0.707107              1j1   1.22474j1.22474|
|  3|1           1.25992           1.44225      0.866025j0.5  1.09112j0.629961  1.24902j0.721125|
|0j1|1 0.769239j_0.638961 0.454832j_0.890577           4.81048  3.70041j_3.07371   2.18796j_4.2841|
|0j2|1 0.940542j_0.339677 0.852887j_0.522096           2.19328 2.06287j_0.745007   1.87062j_1.1451|
|0j3|1 0.973427j_0.228999 0.933693j_0.358074           1.68809 1.64323j_0.386571 1.57616j_0.604461|
+---+-----------------------------------------------------------------------------------------+
```

Exercises

1. Comment on the foregoing tables, including the two-part representation that appears to be used for each complex number.
2. Enter `|@+T c` and comment on the results.
3. Study the tables for other functions such as `-`, `%`, and `^` (and perhaps even `+.` and `*.` and `^.` and `|`).

Two-part representations for individual numbers are not uncommon:

- The result of `36%4` is represented as `9.25`, using an *integer part* and a *fractional part* joined by a dot.
- The result of `23*10^5` can also be represented as `23e5`, using a *factor* and an *exponent* joined by the letter `e`.
- The rational `2%3` can be represented as `2r3`, using a *numerator* and *denominator* joined by the letter `r`.
- Two pi cubed (`2*(o.1)^3`) can be represented as `2p3` using a *factor* and an *exponent* joined by the letter `p`.
- The complex number `3+4*%:_1` is represented as `3j4`, using a *real part* and an *imaginary part* joined by the letter `j`.
- Further cases may be found in the discussion of constants in the J dictionary.

The monadic function `|` used in the table `|@+T a` is called *magnitude*; it yields the square root of the sum of the squares of the real and imaginary parts of an argument. When applied to a real (non-complex) number it is sometimes called the *absolute value*.

Functions defined on real numbers are extended to complex numbers without change, except that they apply to the new element `%:_1` according to the normal rules. The extended functions can therefore be examined in terms of elementary algebra.

B. Addition

The sum of complex numbers can be analyzed in terms of their real and imaginary *components* as follows:

```
   i=:%:_1
   ar=:5 [ ai=:2 [ br=:3 [ bi=:4
   (a=:ar+i*ai),(b=:br+i*bi)
5j2 3j4
```

The following sequence of identities shows that the components of a sum are the sums of the components:

`a+b`	
`(ar+i*ai)+(br+i*bi)`	Definitions of `a` and `b`
`ar+br+(i*ai)+(i*bi)`	Addition is associative and commutative
`(ar+br) + (i*(ai+bi))`	Multiplication by `i` (that is, `i&*`) distributes over `+`

Exercises

4. Enter the foregoing sequence and check that each of the sentences yield the same result.
5. Write and enter a corresponding sequence for multiplication.

C. Multiplication

In discussing multiplication we will use further functions, illustrated as follows:

```
   a=:5j2 [ b=:3j4
   ]ca=:+. a
5 2

   ]cb=:+. b
3 4

   ]ab=:+. a,b
5 2
3 4

   j. 4
0j4

   3 j. 4
3j4

   j./cb
3j4

   j./+.b
3j4
```

Multiplication is analyzed in the following sequence of identities:

```
a*b
(j./ca)*(j./cb)
(ar+j.ai)*(br+j.bi)
(ar*(br+j.bi))+((j.ai)*(br+j.bi))
(ar*br)+(ar*j.bi)+((j.ai)*br)+((j.ai)*j.bi)
(ar*br)+(ar*j.bi)+((j.ai)*br)-(ai*bi)
((ar*br)-(ai*bi))+(ar*j.bi)+((j.ai)*br)
((ar*br)-(ai*bi))+(j.(ar*bi)+(ai*br))
```

Exercises

6. Express the result of the foregoing sequence in English.

   [The real part of a product is the difference of the product of the component lists; the imaginary part is the sum of the real part of each multiplied by the imaginary part of the other.]

7. Re-express the final sentence of the sequence in terms of the table `ab=:+.a,b` .

   [`(-/*/ab)+ (j.+/*/0 1|."0 1 ab)`]

The function `+.` produces a two-element vector representation of a complex argument in terms of its real and imaginary components. If we plot the point whose coordinates are given by `+.` and draw a line to it from the origin we see the possibility of another two-element representation in terms of the length of the line and its angle. This is called a *polar* representation, and is given by the function `*.`. Thus:

```
   *.b
5 0.9272952
```

Angle in radian units rather than degrees

```
   |b
5
```

Magnitude (also called *absolute value* for real arguments)

Multiplication is easily expressed in terms of the polar representation: the magnitude is the product of the magnitudes, and the angle is the sum of the angles. For example:

```
   *. a,b,a*b
5.38516 0.3805064

      5 0.9272952
26.9258    1.3078

   */|a,b
26.9258
```

```
   (+.,*.)a,b,a*b
      5         2
      3         4
      7        26
5.38516 0.3805064
      5 0.9272952
26.9258    1.3078
```

Both representations

The measure of an angle in radians is the length of arc measured on a circle of radius one unit; consequently one-half pi radians is a right-angle, and therefore equivalent to **90** degrees, and pi radians is a "straight" angle of **180** degrees. Since the constant **180p_1** is **180** multiplied by the reciprocal of pi, the conversions between radians and degrees may be expressed as follows:

```
   rfd=:1r180p1&*
```
Radians from degrees

```
   dfr=:180p_1&*
```
Degrees from radians

```
   rfd 0 45 90 180
0 0.7853982 1.5708 3.14159

   dfr rfd 0 45 90 180
0 45 90 180
```

```
   pid=:({.,dfr@{:)"1@*.
```
Polar representation in degrees

```
   pid a,b,0j1,1j1,_1j0
5.38516 21.8014
      5 53.1301
      1      90
1.41421      45
      1     180
```

D. Powers and Roots

We will illustrate the use of powers and roots by developing a function to give the coordinates of regular polygons:

```
   2%:_1
0j1
```
Second (square) root of **_1**

```
   (2%:_1)^i.4
1 0j1 _1 0j_1
```
First four powers of second root of **_1**

```
   +.(2%:_1)^i.4
 1  0
 0  1
_1  0
 0 _1
```
Coordinates of **4**-sided polygon (square)

```
   3%:_1
0.5j0.8660254
```
Cube root of **_1**

```
   +.(3%:_1)^i.6
   1           0
 0.5   0.8660254
_0.5   0.8660254
  _1 1.22461e_16
_0.5  _0.8660254
 0.5  _0.8660254
```
Coordinates of hexagon

```
   reg=:+.@((-:%:_1:)^i.)"0
```
Function for regular polygons

```
   <@reg 3 4 5 6                Boxed polygons of 3-6 sides
+--------------+-----+-------------------+--------------+
|              |     |         1         0|   1          0|
|   1         0| 1  0| 0.309017  0.951057| 0.5  0.866025|
|_0.5  0.866025| 0  1|_0.809017  0.587785|_0.5  0.866025|
|_0.5 _0.866025|_1  0|_0.809017 _0.587785|  _1          0|
|              | 0 _1| 0.309017 _0.951057|_0.5 _0.866025|
|              |     |                   | 0.5 _0.866025|
+--------------+-----+-------------------+--------------+
```

Compare the function **reg** with that used in Chapter 11, and use the plotting functions of that chapter in the following Exercises.

Exercises

8. Plot the figures **(reg 4);(2*reg 4)** in contrasting colors.
9. Use the function **rot** of Chapter 11 to plot rotated figures.

E. Division

Since **%&b** (division by **b**) is the inverse of ***&b** (multiplication by **b**), division is easily expressed in a polar representation: the magnitude is the quotient of the magnitudes, and the angle is the difference of the angles. For example:

```
   a%b
0.92j_0.56

   *.a,b,a%b
5.38516  0.3805064

      5  0.9272952
1.07703 _0.5467888

   %/|a,b
1.07703
```

A complex number may be normalized by dividing it by its magnitude, yielding a complex number with magnitude **1**. For example:

```
   b%5
0.6j0.8

   |b%5
1

   norm=: ]%|
   ]nb=:norm b
0.6j0.8
```

Since a normalized number can be restored by multiplying its norm by a real number, it is often convenient to work in terms of normalized numbers, and then multiply results by appropriate real scale factors.

The reciprocal of a normalized number is simply related to the number itself. For example:

```
   %nb
0.6j_0.8
```

```
   +nb
0.6j_0.8
```

The monad **+** is called the *conjugate*; it reverses the sign of the imaginary part

```
   b*+b
25
```

The product with the conjugate is a real number; the magnitude is its square root.

```
   %:b*+b
5
```

We have yet to examine division in terms of the real/imaginary representation. This may be approached by noting that **a%b** is equivalent to **a*%b** (that is, multiplication by the reciprocal). Since we already have expressions for the product and the reciprocal, the overall result can be obtained by simple, but perhaps tedious, algebra.

Chapter 17

Calculus

A. Secant Slope

If a function `f` is plotted over a range of arguments that includes `x` and `y`, then the straight line through the points `x,f x` and `y,f y` is called a *secant* line, and the quotient of the differences `(f y)-(f x)` and `y-x` is called its *slope*. This slope gives the approximate *rate of change* of the function in the vicinity of `x` and `y`. For example:

```
   f=:*:
   'x y'=:1 3
   rise=:(f y)-(f x)
   run=:y-x
   ]slope=:rise%run
4
```

The secant slope may be expressed in a function that uses the run as the left argument, and in an adverb that may be applied to any function:

```
   ss=:(f@+-f@])%[
   2 ss 1
4

   SS=:1 : '(x.@+-x.@])%['"0
   2 f SS 1
4

   2 ^&3 SS 1          Secant slope of cube with run of 2 at 1
13

   ]r=:10^-i.6
1 0.1 0.01 0.001 0.0001 1e_5

   x=:i.7
   r ^&3 SS/ x          Slopes of cube for various runs and points of
application
     1       7      19      37      61      91     127
  0.01    3.31   12.61   27.91   49.21   76.51  109.81
0.0001  3.0301 12.0601 27.0901 48.1201 75.1501  108.18
  1e_6   3.003  12.006  27.009  48.012  75.015 108.018
  1e_8  3.0003 12.0006 27.0009 48.0012 75.0015 108.002
 1e_10 3.00003 12.0001 27.0001 48.0001 75.0001     108
```

B. Derivative

As the run decreases in size, the slope appears to be approaching a limit, which we may interpret as the *derivative*, the slope of the tangent *at* the point `x`. However, a zero value for the run gives only the meaningless ratio of `0` divided by `0`:

```
   0 ^&3 SS/ x
0 0 0 0 0 0 0
```

For the case of the cube, this derivative may be obtained exactly, because the cube of `x+r` is `(x^3)+(3*(x^2)*r)+(3*x*r^2)+(r^3)`, and the rise (found by subtracting `x^3`) is `(3*(x^2)*r)+(3*x*r^2)+(r^3)`. Dividing this by the run gives `(3*x^2)+(3*x*r)+(r^2)`. Setting `r` to zero in this expression gives `3*x^2`, the derivative of the cube at the point `x`.

The function for the derivative of the cube may therefore be expressed and used as follows:

```
   d3=:3:*^&2
   d3 x
0 3 12 27 48 75 108
```

This result may be compared with the final row of the table of secant slopes.Similar analysis for other powers yields `d4=:4:*^&3` for the derivative of `^&4`, `d5=:5:*^&4` for the derivative of `^&5`, and so on. We define a corresponding adverb for the derivative of any power:

```
   D=:1 : 'x.&*@(]^x."_-1:)'
   2 D x
0 2 4 6 8 10 12

   3 D x
0 3 12 27 48 75 108

   4 D x
0 4 32 108 256 500 864
```

None of this constitutes a proof that the derivatives of all powers follow this pattern, but it does suggest an induction hypothesis for a recursive proof. This matter is treated in Book 3.

Exercises

1. If `f=:^&3` is the cube and `g=:5:*f` is five times the cube, what is the derivative of `g`?

   [Five times the derivative of `f`, that is, `5:*3:*^&2`. Since any secant slope of `g` is five times the slope of `f`, the same is true of the limiting value, that is, the derivative.]

2. If `h=:2:*^&4`, what is the derivative of the sum `s=:g+h`?

[The sum of the derivatives of `g` and `h`,
that is, `(5:*3:*^&2)+(2:*4:*^&3)`]

3. If `C=:3 1 4 2"_` and `E=:0 1 2 3"_` are constant functions, then `t=:+/@(C*]^E)` is a weighted sum of powers. What is its derivative?

   [`der=:+/@(C*E*]^E-1:)"0`. Try `der 1 2 3 4 5`]

C. Polynomials

The preceding **Exercises** developed the fact that the derivative of a weighted sum of powers is itself such a sum, with the exponents decreased by `1`. Since a polynomial is a weighted sum of powers, its derivative is also a polynomial, of degree one less. The derivative of `c&p.` is `d&p.`, where the coefficients `d` are obtained from `c` by applying the following function:

```
   dc=:1:}.]*i.@#
```

For example:

```
   c=:6 5 4 3
   ]i.#c
0 1 2 3

   c*i.#c
0 5 8 9

   1}.c*i.#c
5 8 9

   ]d=:dc c
5 8 9

   (c&p. ,: d&p.) x=:i.7
6 18 56 138 282 506 828
5 22 57 110 181 270 377
```

Exercises

4. Use the fact that the polynomial `0 0 0 1&p.` is equivalent to the cube function to compare the use of the function `dc` with the derivatives of powers obtained in the preceding section.

   [Compare `(dc 0 0 0 1)&p. x=:i.7` with `3 D x`]

5. Comment on the polynomial `(dc dc c)&p.`.

   [This is the second derivative of `c&p.`, that is, the rate of change of the rate of change. For example, if `c&p.` gives the position of a vehicle, then `(dc c)&p.` gives its speed, and `(dc dc c)&p.` gives its acceleration.]

D. Differential Equations

Many important functions are simply related to their rates of change, their first or second derivatives. For example, capital invested at compound interest increases at a rate *proportional* to its value, and the *exponential* or *growth* function (denoted by `^`) increases at a rate *equal* to itself. In other words, the exponential is equal to its derivative.
Is there a polynomial with this property? Clearly not, since the derivative of a polynomial is of lower degree, possessing one less term. However, it is possible to define a power series having the desired property.

Exercises

6. Try to develop a rule or function to generate the coefficients of a power series that equals its derivative.

   [Hint: Begin with the coefficients `c=:1 1 1 1 1 1`,
   and apply the function `dc` to it.]

Pursuing the idea suggested in the exercise we have:

```
   c=:1 1 1 1 1 1
   dc c
1 2 3 4 5
```

Since the second element of the derivative `dc c` is twice the value of the corresponding element of `c`, we replace the third element by one-half its value to compensate:

```
   c=:1 1 1r2 1 1 1
   dc c
1 1 3 4 5
```

Since the third element of `dc c` is now six times its required value of one-half, we replace the fourth element of `c` by `1r6`, and so on:

```
   dc c=:1 1 1r2 1r6 1 1
1 1 0.5 4 5

   dc c=:1 1 1r2 1r6 1r24 1
1 1 0.5 0.1666667 5

   dc c=:1 1 1r2 1r6 1r24 1r120
1 1 0.5 0.1666667 0.04166667

   dc dc c
1 1 0.5 0.1666667
```

It should now be clear that the coefficients are the reciprocal factorials:

```
   ]c=:%!i.6
1 1 0.5 0.1666667 0.04166667 0.008333333
```

```
   dc c
1 1 0.5 0.1666667 0.04166667

   ce=:%@!@i.                    Coefficients for exponential
   ce 6
1 1 0.5 0.1666667 0.04166667 0.008333333

   (ce 10) p. x=:i.4             Ten-term approximation to exponential
1 2.71828 7.38871 20.0634

   ^x
1 2.71828 7.38906 20.0855
```

We have, in effect, defined the exponential as that function which satisfies (i.e., is the solution of) an equation that requires it to equal its own derivative. We may write such equations more clearly in terms of the following *derivative adverb*:

```
   D=:("0)(D.1)                  The scalar first derivative adverb
   ^&3 D                         The derivative of the cube
3&*@(^&2)"0

   ^&3 D x=:i.6                  Applied to an argument
0 3 12 27 48 75

   ^D x                          Derivative of the exponential applied to
argument
1 2.71828 7.38906 20.0855 54.5982 148.413

   (^ = ^D) x                    Test of the differential equation satisfied by ^
1 1 1 1 1 1

   (5&o. = 5&o. D D) x           Sinh equals its second derivative
1 1 1 1 1 1

   (6&o. = 6&o. D D) x           Cosh equals its second derivative
1 1 1 1 1 1

   (1&o. = -@(1&o. D D)) x       Sin is minus its second derivative
1 1 1 1 1 1

   (2&o. = -@(2&o. D D)) x       Cos is minus its second derivative
1 1 1 1 1 1
```

The hyperbolic sine (`5&o.`) and the hyperbolic cosine (`6&o.`) introduced in Chapter 14 both satisfy a similar equation, but one that involves the *second* derivative:

Exercises

7. Use the differential equation satisfied by the hyperbolic cosine together with the approach suggested in Exercise 6 to develop a power series for it.

   [`coshc=:ce*0:=2:|i.`. Use the Taylor series

`6&o. t. i.6` to confirm this solution]

8. Use Taylor series as guides in defining functions to generate power series for the hyperbolic sine, cosine, and sine.
9. Experiment with the weighted Taylor coefficients adverb `t:` for each of the functions treated in Exercises 6-8, study the patterns produced, and state its definition.
10. Predict and confirm the result of `^@- t: i. 10`.
11. Study and experiment with the table of derivatives given in Sec. B, Chapter 2 of Book 3.

E. The Exponential Family

In Chapter 13 we introduced odd and even adverbs that produced the odd and even parts of functions to which they were applied. Moreover, we saw that the odd part of the exponential was equivalent to the hyperbolic sine, and that the even part was equivalent to the hyperbolic cosine. Thus:

```
   O=: .:-
   E=: ..-
   (^O , ^E , ^ ,: ^O+^E) x=:i.6
0  1.1752 3.62686 10.0179 27.2899 74.2032
1 1.54308  3.7622 10.0677 27.3082 74.2099
1 2.71828 7.38906 20.0855 54.5982 148.413
1 2.71828 7.38906 20.0855 54.5982 148.413

   (^O t. ,: ^E t.)x              Coefficients of odd and even parts of
^
0 1   0 0.1666667          0 0.008333333
1 0 0.5         0 0.04166667           0

   (5&o.t. ,: 6&o.t.)x            Coefficients of hyperbolic sine and
cosine
0 1   0 0.1666667          0 0.008333333
1 0 0.5         0 0.04166667           0

   (^O t: ,: ^E t:)x              Weighted Taylor coefficients
0 1 0 1 0 1
1 0 1 0 1 0

   (5&o.t: ,: 6&o.t:)x
0 1 0 1 0 1
1 0 1 0 1 0
```

If `j.` is applied to the argument of the hyperbolic sine (to make it imaginary) , the odd positions of the coefficients of the resulting function `6&o.@j.` are unaffected, because they are all zero. Moreover, those in each fourth place are multiplied by `_1` (that is the fourth power of `j.1`). The function `6&o.@j.` is therefore equivalent to the cosine. Thus:

```
   6&o.@j. t. x
1 0 _0.5 0 0.04166667 0
```

```
   2&o. t. x
1 0 _0.5 0 0.04166667 0
```

The sine may also be similarly expressed in terms of the hyperbolic sine. Moreover all four of these functions can be expressed directly in terms of the exponential, using only the function `j.` and the odd and even adverbs.

Finally, the real and imaginary parts of the function `^@j.` are the cosine and sine respectively. For example:

```
    (+.^@j. x) ; ((cos ,. sin) x)
+-------------------+-------------------+
|         1         0|         1         0|
| 0.540302  0.841471| 0.540302  0.841471|
|_0.416147  0.909297|_0.416147  0.909297|
|_0.989992   0.14112|_0.989992   0.14112|
|_0.653644 _0.756802|_0.653644 _0.756802|
| 0.283662 _0.958924| 0.283662 _0.958924|
+-------------------+-------------------+
```

Exercises

12. Study the plot of sine versus cosine in Section 9J of Book 2.

13. See Chapters 3 (Vector Calculus) and 4 (Difference Calculus) of Book 3.

Chapter 18

Inverses and Equations

A. Inverse Functions

The many scattered references to "inverse" in the index suggests the ubiquity of the notion in math. The general reason for its importance appears in the following example: if we use `heat=:*&4@*:` to compute the output of an electric heater as a function of the voltage applied, we will commonly need the inverse `volts=:%:@(%&4)` to determine what voltage would be required to produce a desired amount of heat. Thus:

```
   heat=:*&4@*:
   volts=:%:@(%&4)
   (];heat;volts@heat) i.5
+---------+------------+---------+
|0 1 2 3 4|0 4 16 36 64|0 1 2 3 4|
+---------+------------+---------+
```

A method for obtaining the inverse of a composition of two functions may be seen in the following example:

```
   cff=:m@s                Celsius from Fahrenheit
     m=:100r180&*          Multiply by conversion factor
     s=:-&32               Subtract a conversion constant
   cff temp=:_40 32 212    Celsius for equal, freezing, boiling points
_40 0 100

   im=:m I=:^:_1           Inverse of m
   is=:s I                 Inverse of s
   m s temp
_40 0 100

   im m s temp
_72 0 180

   is im m s temp
_40 32 212
   ffc=:is@im
   ffc cff temp
_40 32 212
```

```
   cff ffc temp
_40 32 212
```

In general, if several functions are applied one after the other, the inverse is obtained by applying their inverses in reverse order.

Exercises

1. Define the adverb `FI=: f. ^:_1` (fix and invert) and predict and confirm the results of applying it to each of the following functions:

   ```
   cff          m@s          is@im        cff@ffc
   ```

2. Repeat Exercise 1 for the following functions (perhaps using the simpler `I=:^:_1` instead of `FI`):

   ```
   ^            ^.           ^@^.         (^*^@-)
   ```

   [The last function gives a domain error, because `^*^@-` is a constant function (giving `1` for any argument), and a constant function cannot have an inverse.]

3. Repeat Exercise 1 for the following functions:

   ```
   *:           %:           ^&2          ^&3          ^&3@%
   ```

4. Although `*:2` and `*:_2` both yield `4`, the "inverse" function `%:` yields only `2` when applied to `4`. Comment on this matter.

B. Monotonic Functions

A (strictly) monotonic function is one that tends in the same direction as its argument increases. A graph of such a function `f` (as, for example, `f=:^`) provides a visualization of its inverse as follows: at any point `y` on the vertical axis draw a horizontal line to intersect the graph of `f`, and from the point of intersection draw a vertical line to intersect the horizontal axis at `x`. Then `y` is `f x`, and conversely `x` is `f^:_1 y`.

A similar treatment of a non-monotonic function can illuminate the matter raised in Exercise 4: the square function `f=:*:` graphed on a domain that includes both negative and positive arguments is seen to be an even function, and a horizontal line through a point such as `y=:4` intersects the graph in *two* points, giving two possible values for the inverse.

Only a strictly monotonic function can have a proper inverse, but a non-monotonic function may have a useful inverse when restricted to a *principal domain* in which it is monotonic. In the case of the square, the non-negative real numbers provide such a principal domain, and the inverse `*:^:_1` provides the inverse on it.

An inverse for arguments not in a principal domain is often easily obtained from the inverse on the principal domain. In the case of the square it is simply `-@(*:^:_1)`.

Any *periodic* function (such as the sine or cosine) cannot be monotonic, but may be when restricted to a suitable domain.

Exercises

5. Define a function **pn** that gives both positive and negative inverses of the square function, and test it on the argument **x=:0 1 4 9 16 25**.

   [**pn=:(],.-)@%:** (Or use **,:** or **;** instead of **,.**)]

6. Experiment with the functions **n&o.** and their inverses **(-n)&o.** for integer values of **n** from **0** to **8**. Which of the inverses have restricted principal domains?
7. What are the limits of the principal domains of **_1&o.** and **_2&o.**?

   [Apply them to the argument **_1 1**]

C. Under

```
   I=:^:_1
   idr=:10&#.          Inverse of decimal representation; i.e., decimal value
   dr=: idr I          Decimal representation
   dr x=:213
2 1 3

   idr dr x
213

   (];dr;idr@dr) x
+---+-----+---+
|213|2 1 3|213|
+---+-----+---+

   az=:,&0             Append zero
   az dr x
2 1 3 0

   idr az dr x         Decimal value with appended zero
2130

   x*10
2130
```

The foregoing elaborates the familiar idea that a number can be multiplied by ten by appending a zero to its decimal representation. The full expression may be paraphrased in English as "Obtain the decimal representation, append a zero, then evaluate the resulting list in decimal (that is, apply the function inverse to the decimal representation). It illustrates the form **f^:_1 g f** that occurs so often that it is also provided by the conjunction **&.** as follows:

```
   idr@az@dr x
2130

   az&.dr x
2130
```

The general idea is that **f&.g** applies **f** *under* **g**, in the sense that **g** "prepares" the argument for the function **f**, and the "preparation" is finally undone. For example:

```
   +:&.^. y=:4                Double under natural logarithm
16

   *:y                        Is equivalent to squaring
16

   +:&.(10&^.) y=:4
16

   -:&.(10&^.) y=:4
2

   %:y
2
```

Exercises

8. Paraphrase the foregoing expressions in detail.

   [**+:&.^.** takes the natural logarithm of its argument,
   doubles it, and applies the exponential (inverse log).]

9. Experiment with the expressions **6+&.(10&^.)3** and **6-&.(10&^.)3** and comment on the results.

   [The first multiplies its arguments by adding their base-10 logarithms and applying the anti-log (that is, ten-to-the-power); the second uses subtraction to obtain the quotient. The dyadic case of the function **f&.g** is similar to the monadic, but applies the "preparation" function **g** to *each* of the arguments]

10. Define the function **saf=:</** (suppress after first) and experiment with the expressions **saf b** and **saf&.|. b** for various values of the Boolean list **b**, such as **b=: 0 0 1 0 1 1 0**. Comment on the results.

    [**saf** suppresses all ones after the first in a Boolean list; **saf&.|.** suppresses all *before* the *last* by first reversing the list, and again reversing the resulting list after applying the function **saf**.]

D. Equations

A function such as **f=:3 _4 1&p.** may not have a known inverse, but we can obtain the inverse of a given argument such as **y=: 6** by *solving the equation* **y=f x**; that is, by finding a value **x** that satisfies the indicated relation.

If we know values **a** and **b** such that **f** is monotonic in the interval from **a** to **b**, and if **y** lies in the interval from **f a** to **f b**, then a suitable solution x can be obtained by simple repeated approximations: take the average of **a** and **b**; consider the intervals bounded by it and each of them; and choose as a new interval the one whose function values still embrace the argument **y**.

See Section C of Chapter 7 of Book 3 for an executable definition of the foregoing *bisection* method, and Sections D and E for the faster *Newton* and *Kerner* methods that employ derivatives.

The many uses of equations and their solutions in math can mostly be seen as limited means of obtaining inverse functions.

Chapter 19

Readings

A. Introduction

Reading any math text can serve as a stimulus to further exploration, whatever notation it may be expressed in. Those, such as Book 2 and Book 3, that are expressed in J are particularly accessible to users of this book. We will here discuss other books of this type that are easily available because they can be conveniently displayed on the screen (by using the Help menu), and because selections from them can be printed (using the resulting Print menu) for study.

We will here present a few examples from two such books, *J Phrases* and *J Dictionary*.

B. Phrases

After printing the Table of Contents and displaying and reading the first page of the book of *J Phrases* to learn the conventions used, you may choose any chapter for further exploration. Some, such as Chapter 2 (Primitive Notions) and Chapter 8 (Numbers), will provide further elaboration of matters already treated in earlier chapters here. Others, such as Chapters 12 and 13 (Finance and Data) enter new territory.

Chapter 16 (Extended Topics) provides an entree to a wide variety of topics addressed by three authors: C. Burke, D.B. McIntyre, and C. Reiter.

C. Sample Topics

This section of *J Dictionary and Introduction* provides brief treatments of a variety of topics. You might begin with the discussion of Classification and Sets (Sections 8-11), and continue with Directed Graphs and Closure (Sections 20-21). The discussion of polynomials (Sections 23-28) covers some material already treated here in Chapter 14, but also includes matters such as explicit functions for Newton's and Kerner's methods for finding roots, as well as *stopes* that generalize the notions of *falling factorials* and *rising factorials*.

D. Vocabulary and Definitions

Begin by printing out the Vocabulary. Then with the vocabulary displayed, click the mouse on any definition, such as *Self-Classify* . *Equal* in the upper left corner.

A study of the defintion will probably provide all the information you need concerning the conventions used. If not, display the page of the dictionary headed by *III. Definitions* for details of them.

References

1. Reiter, Clifford A., *Fractals, Visualization, and J*, Second Edition, Jsoftware, 2000.
2. Iverson, Kenneth E., *Concrete Math Companion*, Jsoftware, 1995.
3. Graham, Ronald L., Donald E. Knuth, Oren Patashnik, *Concrete Mathematics*, Addison Wesley, 1989.
4. Lakatos, Imre, *Proofs and Refutations: the logic of mathematical discovery*, Cambridge University Press, 1976
5. Hui, Roger K.W., and Kenneth E. Iverson, *J Dictionary*, Jsoftware, 1998.

   This text is available on-line in the J system, as discussed in Chapter 10.
6. *American Heritage Dictionary of the English Language*, Houghton-Mifflin. (Any edition that includes the appendix of Indo-European roots.)
7. Thomas, Lewis, *et cetera, et cetera : Notes of a Word-Watcher*, Little, Brown and Company, 1990.

Index

absolute value, 102, 104

addition, 1, 5, 9, 18, 24, 44, 46, 56, 57, 75, 76, 80, 87, 100

Adverbs, 23, 27

agenda, 50, 53

alphabet, 28

ambivalent, 24

anagram, 10, 12, 83

and, 20

appendices, 7

approximating functions, 90

area transformation, 80

arguments, 6, 18, 23, 24, 31, 34, 38, 46, 50, 69, 78, 79, 87, 90, 94, 97, 104, 109

arithmetic, 7, 33, 57, 58, 97

Arithmetic, 97

arithmetic progression, 57

array, 29, 30, 34

assign, 5, 87

associative, 57, 103

associativity, 57, 75

atom, 34, 50

atop, 5

average, 31, 33, 99

axes, 30, 31

axioms, 14

axis, 31, 91

backspace, 5

base-8, 37, 40

base-value, 37, 38

bisection, 121

block, 40, 41

Boole, 20

Boolean, 120

Bordered, 18, 101

box, 13, 30, 64, 69, 70, 84

boxed roots, 88

Boxing, 32

by, 24, 101

calculus, 7

Calculus, 109

carries, 34, 39, 40, 44

catenate, 27

Celsius, 117

Classification, 122

Closure, 122

Coefficients Representation, 87

Colors, 67

commutative, 18, 26, 57, 103

commutativity, 57, 75

commuted, 20

companion volume, 7

Comparisons, 2

complex numbers, 4, 70, 89, 97, 98, 99, 100, 102

Complex Numbers, 99, 101

composition, 25, 89, 117

Concrete Math Companion, 7, 91

conjunction, 7, 25, 33, 34, 50, 60, 76, 78, 96, 98

conjunctions, 27, 76

constant, 27, 29, 34, 50, 51, 57, 105, 111

constant function, 118

constants, 27, 59, 102

conventional notation, 7

conventions, 123

coordinate geometry, 65

coordinate system, 65

Coordinates, 65

copula, 25, 28, 42

copulative conjunction, 25

cosine, 89, 90, 91, 92, 113, 114, 115

counting numbers, 97, 98

cube, 3, 33, 59, 109, 110, 111, 113

cursor, 5, 64

Data, 122

Decimal, 37

decimal point, 28

Decimal representation, 119

deductive, 14

Definitions, 123

degenerate triangle, 80

degrees, 104, 105

delete, 5

denominator, 102

derivative, 89, 110, 111, 112, 113

Derivative, 110

dervatives, 89

determinant, 68, 80

determinants, 69

dfr, 105

diagonally, 44

Difference Calculus, 115

Differential Equations, 112

Directed Graphs, 122

Displacements, 66

distributes, 56, 75, 76, 80, 103

Distributivity, 75

divided by, 24, 110

Division, 106

dropping the menus, 63

Editing, 64

English, 5, 10, 11, 12, 23, 24, 25, 27, 28, 31, 33, 34, 40, 104, 125

Equal, 123

equals, 2, 3, 4, 56, 57, 59, 89, 91, 92, 112, 113

Equations, 120

erase, 5

etymology, 34, 66

even, 1, 3, 9, 10, 14, 18, 23, 68, 79, 91, 92, 97, 102, 114, 115

executable, 121

execution, 24, 26, 27, 50, 64

exploration, 1, 2, 4, 7, 14, 63

exponent, 102

exponential, 24, 89, 90, 91, 112, 113, 114, 115

Exponential Family, 114

Extended Topics, 122

factor, 80, 102

factorial, 12, 19, 24, 49, 89

Fahrenheit, 117

falling factorials, 122

false, 20

Fibonacci series, 91

Finance, 122

Fractals, 7, 125

fractional part, 102

function, 2, 3, 5, 6, 10, 12, 13, 19, 24, 25, 27, 30, 31, 34, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 58, 59, 60, 61, 66, 68, 75, 76, 77, 78, 79, 80, 81, 83, 84, 85, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 102, 104, 105, 106, 109, 110, 111, 112, 113, 114, 115

Function Tables, 17

generalized inverse, 95

geometric figures, 65, 80

geometric progression, 58

gerund, 50, 51

gopen, 67, 70

gpolygon, 67

grammar, 23, 24, 27, 63

Grammar, 23, 24

graph, 65, 66, 67, 68, 89, 91

Greatest Common Divisor, 20

guesses, 14

Help, 122

help menu, 63, 84

heron, 66

Heron's formula, 66, 68

hierarchical rules, 24

hyperbolic, 92, 113, 114, 115

identities, 103, 104

identity, 11, 49, 56, 79, 90

Identity, 5, 98

imaginary, 92, 100, 101, 102, 104, 107, 114, 115

improve, 14, 28, 95

indexing, 52, 83

Indo-European, 35, 125

induction hypothesis, 59, 110

induction hypothesis), 59

INDUCTIVE PROOF, 59

Infinite rank, 50

Inner Product, 78

integer part, 102

integers, 3, 9, 12, 40, 49, 59, 65, 98, 99

interpreted, 24, 51, 52

intervals, 120

inverse, 10, 38, 51, 78, 93, 95, 97, 98, 99, 106

Inverse, 117

Inverses, 117

INVERSES, 97

Irrational Numbers, 99

irrationals, 98, 99

iteration, 40, 41

J, 2, 7, 23, 24, 25, 27, 28, 33, 34, 40, 63, 64, 67, 87, 92, 98, 102, 125

J Introduction and Dictionary, 122

J Phrases, 122

Kerner, 121

Kerner's, 122

Lakatos, 14, 56, 57, 125

laminate, 27, 90

languages, 14

Least Common Multiple, 20

left to right, 26, 27, 65

`length`, 26, 66, 99, 104, 105

Lewis Thomas, 34

Linear functions, 76

Linear Functions, 75

Linear Vector Functions, 77

Linearity, 76, 92

link, 13, 27

list, 2, 3, 5, 6, 10, 11, 12, 17, 18, 27, 28, 29, 34, 38, 40, 42, 43, 44, 46, 56, 57, 60, 87, 88, 90, 96

logic, 7, 14, 97, 125

magnitude, 46, 90, 102, 104, 106, 107

major cells, 31

math, 1, 2, 4, 5, 6, 7, 9, 24, 33, 34, 63, 76, 89

Math, 9

mathematical, 12, 125

matrix, 34, 60, 78, 79, 81, 85, 92, 93, 94, 96

matrix product, 78

mean, 31, 32, 33, 34, 39, 58

Mixed Bases, 47

MN, 23, 24, 33, 34, 87

monomial, 87

Monotonic, 118

multiplication, 1, 5, 9, 17, 18, 24, 26, 44, 56, 87, 90, 98, 99, 100, 103, 106, 107

Multiplication, 44, 103

native language, 23

natural logarithm, 120

negative numbers, with the standard form limited (as it is for positive arguments) to numbers, 46

Newton, 121

Newton's, 122

normalization, 40, 44, 47

normalized number, 107

noun, 9, 25, 34, 50

nouns, 24, 25, 28

Nouns, 23

number of *items*, 12, 31, 40, 76, 79, 90, 94

Number of items, 31

Numbers, 122

numerator, 102

oblique, 44, 45

octal, 37

odd numbers, 3, 5, 6, 51, 55

operator, 34, 51

or, 20

origin, 65, 66, 91, 104

over, 101

Padding, 44

Parity, 91

parse, 24, 25

parsed, 24

pattern, 6, 56, 110

patterns, 4, 5, 14, 114

pentagon, 69

perform, 34, 39, 40, 52

perimeter, 26, 66

periodic function, 118

permutation, 9, 10, 11, 52, 56, 81, 83, 84, 85, 98

permutations, 7, 10, 11, 12, 52, 70, 84, 97, 98

permuted, 10, 12

permuting, 3

Phrases, 122

pi, 33, 34, 102, 105

Plotting, 69

PLOTTING, 70

polar representation, 104, 106

polygons, 66, 68, 70, 105, 106

polynomial, 60, 61, 87, 88, 89, 90, 91, 92, 93, 94, 95, 111, 112

Polynomial Approximations, 94

polynomials, 7, 24, 87, 88, 89, 91, 97, 122

Polynomials, 87

POLYNOMIALS, 111

power, 1, 3, 24, 87, 89, 90, 91, 92, 93, 95, 98, 110, 112, 113, 114

Power, 19

power series, 89, 112

POWERS AND ROOTS, 105

predecessor, 97, 98

Primitive Notions, 122

principal domain, 118

Pro-adverb, 28, 33

Pronoun, 28, 33

Pronouns, 23

proof, 14, 55, 56, 57, 58, 59, 60, 61, 110

proofs, 7, 9, 13, 14, 55, 58, 60, 61

Proofs, 13, 55

proper inverse, 98, 99

proposition, 57

Proverb, 28, 33

punctuation, 24

Punctuation, 25

Pythagoras, 66, 99

quadrant, 66

quadrants, 18

quotient, 106, 109

radian units, 104

radians, 105

Ramble, 6

random, 3

random generator, 29

rank conjunction, 50

rank-**k**, 31

ranks, 32

rate of change, 89, 109, 111

Rational Numbers, 98

rationals, 98, 99, 100

Readings, 122

reciprocal, 24, 89, 105, 107, 112

Recursion, 49

recursive proof, 110

Refutations, 14, 125

reg, 70, 105, 106

relation, 1, 9, 12, 34, 35, 42, 55, 75

relations, 9, 10, 12, 14, 34

Relations, 9

repeatable, 3

repeated approximations, 120

report, 29, 30, 31, 32, 34

Reports, 29

representation, 28, 37, 38, 39, 40, 43, 45, 46, 68, 79, 83, 84, 87, 88, 102, 104, 105, 106, 107

Representations Of Functions, 83

Research, 6

reversal, 38

Reverse, 31

rfd, 68, 105

right parenthesis, 41

right to left, 26

rising factorials, 122

Roots Representation, 88

Roots table, 102

Rotate, 31, 66

SAMPLE TOPICS, 122

Save As, 64

scan, 25

script, 59, 64, 68

Script Windows, 64

secant line, 109

Secant Slope, 109

second derivative, 113

selection, 52

Self-Classify, 123

sets, 7

Sets, 122

Shape, 31

signum, 46, 51

similar, 3, 9, 10, 35, 98, 113

sine, 89, 90, 91, 92, 113, 114, 115

slope, 89, 109, 110

solving, 120

sort, 3, 4, 9, 10, 13

specific arguments, 34

Spelling, 23, 28

square, 3, 118

squares, 6, 59, 95, 102

stitch, 27

stopping condition, 49

stopping value, 49

subtotals, 6, 78

Subtraction, 46

successor, 97, 98

sum, 6, 24, 25, 29, 39, 40, 43, 51, 55, 56, 57, 58, 59, 60, 61, 75, 77, 78, 87, 89, 91, 102, 103, 104, 110, 111

sum function, 6, 25

symmetric, 4, 18, 56, 57, 65

table, 13, 17, 18, 19, 24, 34, 43, 44, 45, 52, 68, 69, 78, 79, 81, 84, 88, 92, 93, 101, 102, 104, 110, 114

TABLE, 18, 19, 20, 23

tables, 5, 13, 17, 18, 20, 30, 52, 76, 95, 101, 102

tangent, 89, 110

tangible representations, 14

Taylor adverb, 61

Taylor series, 90, 91, 94, 95, 113, 114

Taylor series adverb, 90

Terminology, 33

tetrahedron, 81

the reciprocal factorials, 112

ties, 50

Tools, 63

Tower of Hanoi, 51

transcendental functions, 89

transformations, 14

translating, 7

transpose, 13, 18

TRANSPOSITION, 31

tree, 34, 35

true, 20, 23, 34, 35, 55, 57, 88, 110

Two-part representations, 102

Under, 119

under open, 69

Vandermonde matrices, 94

Vandermonde matrix, 93, 94

variable, 34

vector, 34, 66, 77, 78, 79, 80, 81, 83, 104

Vector Calculus, 115

verbs, 5, 23, 25, 27, 28

Versatility, 89

Visualization, 7, 65, 68, 125

Vocabulary, 123

volume, 7, 81

weighted sum, 77, 111

Weighted Taylor coefficients, 114

whole numbers, 3, 7, 97

Width, 25, 26

window, 59, 64, 67, 68

word-formation, 28