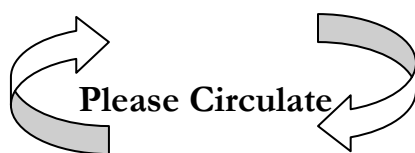
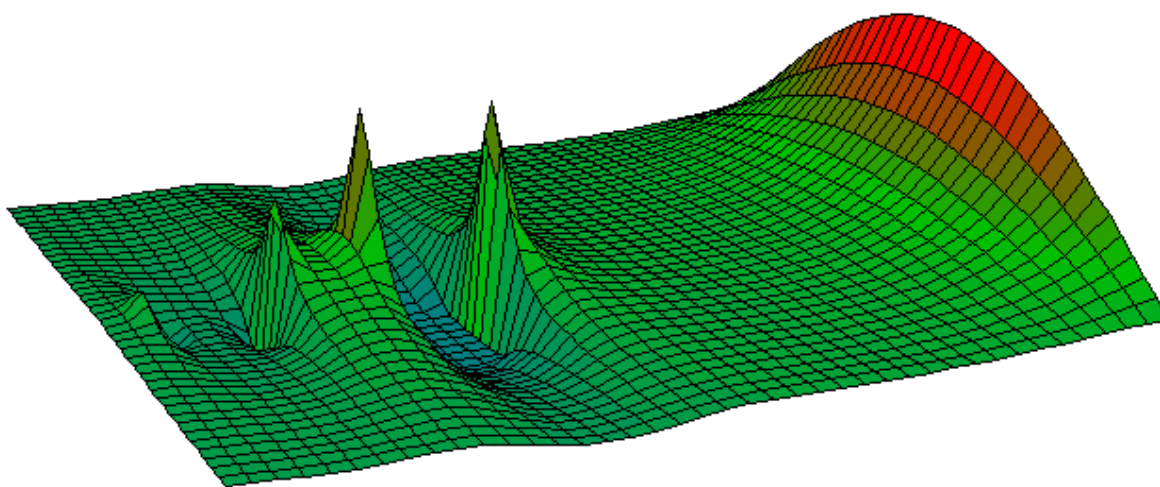


# *Journal of* J

*An interdisciplinary journal on J programming language and applications in science*



Journal of J. Vol.1, No. 1, January 2012

MPM press

ISSN: 2174-9280

A open access Journal

Vol.1, No.1 February 2012

# Journal of J (J2-team)

**J, a language for the science**

## **Aims and scope**

**Journal of J** is a **non professional and nonprofit** journal in science, involving a large research and users community in J programming language.

**Journal of J** is an interdisciplinary **journal** devoted to J and science.

**Journal of J** aims to provide fast access to papers about science and J.

**Journal of J** embodies the following principle:

*Open Access:* Knowledge is a public good. All readers have [open access to reading and downloading papers](#). The simple and free access ensures maximum readership and high citation records for published papers.

**Contact:** journalofj at hotmail dot com

## **Style and Contents**

**Journal of J** aims to cover all the main areas of science. Inevitably, articles in different areas are addressed at different audiences. Many of the articles submitted to the journal are standard technical pieces, addressed to a purely academic audience

To attract this variety of contributions **Journal of J** will contain the following areas:

- Mathematics: number theory, logic, calculus, algebra, arithmetic, algorithms and others...
- Physics: dynamical systems, chaos, fractals, disorder, statistical physics and others...
- Computer science, Visualization, Engineering, Computer Art, ...

**Visit:**

<http://sites.google.com/site/jforscience/>

**Edited by Mikel Paternain**

## Editorial

Este primer número del nuevo año recogemos interesantes colaboraciones centradas en aplicaciones de J.

J es un lenguaje de programación extremadamente potente como se puede apreciar por ejemplo de este número sobre la aplicación del mismo al mundo del *trading* financiero. El equipo de *Journal of J* sigue confiando en el proyecto y esperamos

que esta revista sirva como punto de encuentro de todos los entusiastas en J. De igual manera, esperamos que las interesantes contribuciones de los programadores sirvan como lección para extender este lenguaje de programación.

Esperamos que el nivel de las contribuciones siga aumentando a lo largo de este nuevo año.

Mikel Paternain (Editor)

mikelpater at Hotmail dot es

## J for trading

**Thomas Costigliola**

January 9, 2011

J is an interesting programming language in the sense that is well suited and even fun for solving small problems yet, in my experience, scales well to large data processing applications. I have been using J at my present job for over 5 years developing and maintaining a general quantitative trading system. At the company J code is responsible for a multitude of tasks including downloading market data, transforming and moving data in and out of databases, implementing a quantitative statistical model, generating reports and interactive GUIs, interfacing with trading systems, real-time plotting, performance tracking and risk management tools, and general business automation tasks. It is apparent that J has proven to be a very versatile language.

Because it is "high level" it allows us to prototype new features quickly but it is also robust enough to implement production quality code that is easily readable and maintainable. The latter is partly facilitated by allowing the programmer to concentrate on the high level operations necessary to solve a problem rather than figuring out the sequence of computer instructions needed to implement such operations. This should sound familiar to functional programmers; in fact if you use the "tacit" coding style in J you are programming functionally.

There some features, though, that set it apart from other functional languages. It has quite a terse syntax and rich set of primitive operators, both of which sometimes confuse beginners,

however, once mastered can lead to clean, concise and enjoyable code. The terseness of the syntax does not take away from its flexibility though. The use of strategically named verbs, adverbs and conjunctions allows for "mnemonic programming", or building domain specific micro-languages. The J interpreter is also fast, in many tasks achieving speeds near much lower level languages with far less effort.

What follows is an introduction to some of the features mentioned above that we have found useful at the company.

Reading and writing data from and to files is painless. A literal array can be written directly to a file and then read back. Let's start with some basic definitions.

```
read=. 1!:1
write=. 1!:2
tonumbers=. ".
tochars=. ":
```

The following generates 100 random integers between 0 and 99, writes them to a file, reads it back and compares them to the original.

```
NUMBERS=. ?100#100
(tochars NUMBERS) write <'data.txt'
DATA=. tonumbers read <'data.txt'
DATA -: NUMBERS
```

Now let's try to do some interesting things with these numbers. I have given mnemonics even to some primitive verbs to demonstrate how elegantly and naturally the language can flow.

```
sum=. +/
count=. #
mean=. sum % count
variance=. mean@:*: - *:@:mean
stdev=. %: @ variance
rolling=. 2 : 'v\' NB. Ignoring the left parameter for
mnemonics.
```

```
mean DATA
```

```
54.73
```

```
stdev DATA
```

```
27.9574
```

Notice how the 'mean' is the 'sum divided by the count' and the 'variance' is the familiar 'mean of square minus the square of mean'. J allows us to express the operations how we understand them rather than worry about loops, counters, memory etc. Here is a further example again demonstrating the elegance of mnemonics and the code reusability of adverbs and conjunctions.

```
20 observation rolling mean DATA
```

```
20 observation rolling stdev DATA
```

```
20 observation rolling (mean % stdev) DATA
```

Looking at all of those numbers is boring. Let's see if we can plot the results instead.

```
load 'plot'
```

```
pd 'reset'
```

```
pd 20 observation rolling mean DATA
```

```
pd 20 observation rolling stdev DATA
```

```
pd 20 observation rolling (mean % stdev) DATA
```

```
pd 'show'
```



Piece of cake!

Working with relational databases is also a snap. J uses the ODBC system where most popular database systems provide drivers for most operating systems. Assume you have set up an ODBC DSN called 'mydatabase' on your system

```
load 'dd'

CON=. ddcon 'dsn=mydatabase'

DATA=. ddfet ('select * from mytable' ddsel CON) , _1

dddis CON

_1
```

That is all the code needed query columns of data into J.

Finally, no matter how good your programming language is there will come a time when you will want to, or need to use an external library. J even makes this easy with its `cd` (call dynamic-library) interface. The verb '`cd`' takes as a left argument the external function and its prototype you want to call and as its right argument, the parameters you wish to pass to this function. The following is an example of using the popular `cURL` library from J. `cURL` allows your program to use various internet protocols and is available for most operating systems. I will use it to download data from Yahoo Finance. Note that this code was written and run on Linux, the `_only_` potential change necessary to run on a different OS is the location of `libcurl` itself. For cross platform code the location can be dynamically coded by checking the global variables `IFUNIX` and `IFWIN`; this was left out for clarity and brevity.

First, let's define the `cURL` functions we will use and assign them to J verbs by binding the appropriate left argument to `cd`. A couple `cURL` internal constants are defined as well.

```
curl_init=.    [:{'libcurl.so curl_easy_init      *      ' & cd
curl_cleanup=.  'libcurl.so curl_easy_cleanup    n *      ' & cd
curl_setopt=.   'libcurl.so curl_easy_setopt     n * i *' & cd
curl_perform=.  'libcurl.so curl_easy_perform    i *      ' & cd

CURLOPT_URL=.  10002

CURLOPT_FILE=. 10001
```

Notice the structure of the literal left argument to `cd`:

```
dll_name function_name return_type param_1_type  param_2_type
... param_n_type
```

Here, 'n' means void return type, '\*' means pointer type and 'i' means integer type. The J interpreter handles converting back and forth between J types and C types. See the section "Calling DLL's" in the J dictionary for more details.

Next we write a verb that will use curl to download daily historical data, in csv format for some stock indices.

```
open_file= 1!:21

close_file=. 1!:22

yahoo_url=.
'http://ichart.finance.yahoo.com/table.csv?=%5E', , '&d=0&e=9&f=201
2&g=d&a=11&b=23&c=1980&ignore=.csv'

download_history=. verb define

    F=. open_file <y

NB. Create a file with the same name as the symbol

    C=. curl_init ''

    curl_setopt C;CURLOPT_URL;yahoo_url y

NB. Set the URL curl will fetch

    curl_setopt C;CURLOPT_FILE;<<F

NB. Tell curl which file to download to

    curl_perform <C

NB. Perform the HTTP request

    curl_cleanup <C

    close_file F

NB. Close the file

)
```

The verb 'download\_history' is an example of an explicit definition. It essentially wraps a sequence of J instructions you wish to execute recurringly with different parameters. Let's use it to download historical prices for the S&P, Dow Jones and NASDAQ indices.

```
SYMBOLS=. 'GSPC';'DJA';'IXIC'

download_history each SYMBOLS
```

Now just a few more definitions to do something interesting with this data. It should all be understandable through the mnemonics.

```
append_comma=. ,&' ,'  
cut=. ;._2  
cut_csv=. [:}. < cut @ append_comma cut  
fetch=. &{::  
eachrow=. "1  
get_closing_price=. tonumbers @: (4 fetch eachrow)
```

NB. The closing price is the 4th column of data

```
log=. ^.  
return=. log @ %/ NB. Continuously compounded return
```

```
RETURNS=. (2 observation rolling return ]) @  
get_closing_price_pricing @ cut_csv @ read SYMBOLS
```

Finally, plot the 100 day rolling standard deviation of each index.

```
plot 100 observation rolling stdev eachrow RETURNS
```

All of the code in a single script with some enhancements is available on my page on the J wiki (<http://www.jsoftware.com/jwiki/ThomasCostigliola>).



# Letters to Editor

## Bo Jacoby

NB. Consider a lottery with 10000 tickets.

NB. There is one big prize, ten medium prizes,

NB. one hundred tiny prizes,

NB. and the rest of the tickets are blanks.

NB. You you want to know about a SAMPLE,

NB. based on the POPULATION

    ] p = . (,10000-+/) 1 10 100  
1 10 100 9889

NB. This is called DEDUCTION.

NB. Buying no tickets, what do you get?

    0 deduc p

0 0 0 0

0 0 0 0

NB. You get exactly nothing, of course.

NB. Buying one ticket, what do you get?

    1 deduc p

    0.0001      0.001          0.01  
0.9889

0.0099995 0.031607 0.0994987  
0.10477

NB. You get 0.01% chance for winning the big prize,

NB. 0.1% chance to win a medium prize,

NB. 1% chance for winning a very small prize,

NB. and 98.9% chance for the ticket to be blank.

NB. The number of blanks is either one or zero,

NB. but the mean number of blanks is 0.9889,

NB. and the standard deviation is 0.10477.

NB. So you get 1 blank, give or take 0.1.

NB. Buying ten tickets, what do you get?

    10 deduc p

    0.001          0.01          0.1

9.889

0.031607 0.099905 0.314501

0.331163

NB. You get 0.1% chance for winning the big prize.

NB. You get 1% chance for winning a medium prize.

NB. You get 10% chance for winning a tiny prize.

NB. You get 9 blanks, give or take 1.

NB. Buying one hundred tickets, what do you get?

    100 deduc p

    0.01          0.1          1      98.89

0.0994987 0.314501 0.99005 1.0425

NB. You get 1% chance for winning the big prize.

NB. You get 10% chance for winning a medium prize.

NB. You get 1 tiny prize, give or take 1.

NB. You get 99 blanks, give or take 1.

NB. Buying one thousand tickets, what do you get?

    1000 deduc p

    0.1          1          10      988.9

0.3 0.948256 2.98511 3.14326

NB. You get 10% chance for winning the big prize.

NB. You get 1 medium prize, give or take 1.

NB. You get 10 tiny prize, give or take 3.

NB. You get 989 blanks, give or take 3.

NB. Buying all ten thousand tickets, what do you get?

    10000 deduc p

1 10 100 9889

0 0 0 0

NB. You get of course exactly all the prizes

NB. and all the blanks too.

NB. The deduction formula

    deduc

\*`%`:3"2@(:, (: %:@\* -.))@((, : ,  
1:) % +/@])

NB. computes mean values and standard deviations for

NB. hypergeometric distributions.

NB.

=====

NB. The dual case is when you only know the

NB. number of tickets in the lottery, but not how

NB. many prizes of each size there are.

NB. You you want to know about the POPULATION,

NB. based on the SAMPLE.

NB. This is called INDUCTION.

NB. The induction formula

induc

(T@}: , }.)@(T~ deduc T)

NB. is my modest gift to mankind.

NB. It involves the simple transformation

T

-@(+ #)

NB. Buying all the tickets in the lottery,

NB. what do you know about the lottery?

p induc 10000

1 10 100 9889

0 0 0 0

NB. You know that the lottery is exactly:

NB. one big prize, ten medium prizes,

NB. and one hundred tiny prizes,

the rest being blanks.

NB. Buying 1000 tickets and

getting 1 medium prize,

NB. 10 tiny prizes, and 989 blanks,

NB. what do you know about the lottery?

0 1 10 989 induc 10000

8.96414 18.9283 108.606 9863.5

9.44152 13.3457 31.1575 35.0972

NB. The actual number is almost certainly no more than

NB. 3 standard deviations above the mean value.

({.+3\*{:})0 1 10 989 induc

10000

37.2887 58.9653 202.078 9968.79

NB. There are almost certainly no more than 37 big prizes

NB. and no more than 59 medium prizes

NB. and no more than 202 tiny prizes

NB. and no more than 9969 blanks.

NB. The actual number is almost certainly no less than

NB. 3 standard deviations below the mean value.

({.-3\*{:})0 1 10 989 induc

10000

\_19.3604 \_21.1087 15.1331 9758.21

NB. There are almost certainly no less than 15 tiny prizes

NB. and no less than 9758 blanks.

# Q con ejemplos

Traducido y adaptado por

Mikel Paternain

Mikelpater at Hotmail dot com

©2011

## 1. Introducción

Q es un lenguaje propietario de *programación matricial* diseñado y puesto a punto por Arthur Whitney, y comercializado por Kx Systems.

Este documento es una traducción y adaptación del documento **Q by example** y que, junto con una versión de evaluación del intérprete Q, está disponible de forma gratuita en la web de Kx System. Lo ideal para seguir este documento es tener instalado el intérprete e introducir todos los ejemplos en la consola de Q.

Nuestro objetivo es facilitar la introducción a este interesante lenguaje pero no es una guía definitiva de programación en Q. Aquellos lectores interesados en adquirir profundos conocimientos de Q deberán leer el libro *Q for Mortals* (QFM) de Jeffrey A. Borror. Hemos utilizado QFM ampliamente como referencia para las explicaciones previas antes de introducir los ejemplos.

## 2. Tipos

Q es un lenguaje donde todos los datos se construyen en base a *Átomos*. Un átomo es un valor correspondiente a un tipo de dato específico. Q soporta varios tipos algunos de los cuales se muestran en la tabla inferior. Para otros tipos de datos y más información consúltase QFM.

Tipo	Tamaño	Carácter	Número	Notación	Valor nulo
boolean	1	b	1	1b	0b
byte	1	x	4	0x26	0x00
short	2	h	5	42h	0Nh
int	4	i	6	42	0N
long	8	j	7	42j	0Nj
real	4	e	8	4.2e	0Ne
float	8	f	9	4.4	0n
char	1	c	10	"z"	" "
symbol	*	s	11	`zaphod	`
month	4	m	13	2006.07m	0Nm

### 3. Listas, tablas y diccionarios

Uno de los aspectos más potentes de Q es el tratamiento de las listas, tablas, arrays y diccionarios.

Para construir y definir una lista, “encerramos” los elementos entre paréntesis y los separamos por punto y coma, es decir:

```
q)L1: (1,3,5,7,9)           / Lista formada por cinco enteros  
                             impares.
```

```
q)L2:(2;-19.3;2.34e;0b;"a";`s) / Lista formada por 6 datos de  
                             diferente tipo
```

Obsérvese como la asignación de una lista de datos a una variable (L1 y L2 en el ejemplo anterior) se realiza mediante el empleo de la primitiva :

Podemos utilizar una notación simplificada también eliminando los paréntesis y el punto y coma, separando los elementos por espacios en blanco

```
q)L3: 12 34 34 23 66           /lista de 5 enteros. Asignación de la  
                             lista a la variable L3
```

Debemos destacar que las listas en Q se ordenan de izquierda a derecha teniendo el primer elemento índice cero. Es decir, una lista de n elementos es recorrida por índices que varía desde cero hasta n-1.

Para extraer un elemento de una lista procedemos mediante el empleo de **nombre[i]**. Basándonos en una de las listas anteriores tenemos por ejemplo

```
q)L2[4]
```

```
"a"
```

Podemos alterar/modificar un elemento de una lista mediante la siguiente instrucción **nombre[i]:valor** Siguiendo con el ejemplo anterior

```
q)L2[3]:1b /reemplaza el elemento 3 con valor 0b por 1b
```

Podemos incrementar la complejidad de los datos tratados considerando “listas de listas” tales como,

```
q)M:( 0;1;2;(3;5;7);(2;3))
```

```
q)M
```

```
1           / elemento con índice 0
```

```
2           /idem. Con índice 1
```

3 5 7            /idem. Con índice 2. Se trata de una lista de tres elementos

2 3            /idem. Con índice 3

La lista M se compone de dos elementos simples y dos sub-listas, una de tres elementos y otra de dos.

Para extraer un elemento dentro de una sub-lista podemos emplear **nombre[i] [j]** para extraer el elemento j de la sub-lista i. Por ejemplo:

```
q)M[2][2]
```

7

```
q)M[3][1]
```

3

```
q)M [3][3]
```

0N            / al indexar de forma incorrecta Q responde con elemento nulo

Podemos alterar el valor de un elemento de la lista mediante el empleo del indexado y la asignación, es decir, `nombre[i]:valor`. Por ejemplo:

```
q)K:(1;3;5;7;9)    / define una lista compuesta por cinco números
```

```
q)K[3]:0            / asigna el valor 0 al elemento con índice tres
```

```
q)K
```

1 3 5 0 9            / recuérdese que el índice comienza por cero

Este tipo de asignación puede hacerse con varios elementos:

```
q)K[0 1 2]:1 1 1    / reemplaza el valor de los elementos con índice 0, 1,2
```

```
q)K
```

1 1 1 0 9

En Q denominamos *diccionario* a una colección ordenada de pares de valores. También se denomina *mapa o asociación* ya que define una asociación explícita entre entradas y salidas mediante una correspondencia posicional entre una *lista dominio* y una *lista rango*. Se construyen los diccionarios mediante el empleo de la primitiva !

Veamos un ejemplo:

```
q)d:`a`b`c`d`e`f!12 23 23 23 12          /define el diccionario
q)count d                                   /cuenta el número de pares
5
q)Key d
`a`b`c`d`e
q)Value d
12 23 23 23 12
q)d
a | 12
b | 23
c | 23
d | 23
e | 12
```

Al tratarse de una introducción no podemos extendernos demasiado en este punto. Recomendamos la lectura del capítulo 6 de QFM para conocer más sobre el concepto de diccionario.

Otro concepto de mucha importancia en Q son las tablas. De hecho, las tablas son la base de kdb+ (base de datos de alto rendimiento asociada a Q) Debido a la importancia y amplitud de aspectos implicados en el tratamiento de tablas recomendamos la lectura del capítulo 7 de QFM.

#### 4. Primitivas, verbos, adverbios, conjunciones y funciones

Q dispone de varias funciones (acciones sobre datos) básicas tales como la suma +, la resta -, pero también otras como `count` para contar los elementos de una lista, tabla o diccionario, `reverse` para invertir una lista, etc. Nuevamente remitimos al lector a QFM para conocer en profundidad todas las funciones definidas en Q. Nos interesa aquí explicar la forma que en el usuario puede definir sus propias funciones en base a las reglas del lenguaje.

Para definir una función usaremos en Q el siguiente esquema:

$$\{[p1;p2;p3; \dots ;pn] \ e1;e2;e3 \dots ;en\}$$

Donde  $p_1; p_2; p_3; \dots; p_n$  son parámetros formales de la función y  $e_1; e_2; e_3 \dots$   $; en$  es una secuencia de expresiones que se evalúan de izquierda a derecha. Veamos algunos ejemplos:

```
q)f : {[x]x+til x}
```

```
q)f[3]
```

```
3 4 5
```

Analicemos con detalle la expresión anterior. En primer lugar vemos que se emplea la primitiva `til n` que genera una lista de enteros desde cero hasta  $n-1$ . A la función se le asigna el nombre `f` y entre corchetes `{}` se escribe la función. En este caso la función toma un único argumento  $x$ , escrito entre paréntesis `[]`, y devuelve el resultado de sumar el valor de  $x$  a los elementos de la lista de enteros desde 0 hasta  $n-1$ .

Podemos obviar la declaración del argumento dentro de la función y asignar directamente la expresión al nombre de la función

```
q)g : {x+cos x}
```

Si la función posee más de un argumento, estos se declaran también dentro de la función,

```
q)h:{[x;y;z]x+y+til z}
```

```
q)h[5;2;10]
```

```
7 8 9 10 11 12 13 14 15 16
```

Una función también puede definirse sin hacer una asignación a una variable. Tal función se denomina *anónima*. Por ejemplo:

```
q){x+y}[5;65]
```

```
70
```

Q dispone de una amplia colección de funciones básicas llamadas *verbos* con las que, además de obtener resultados directos al actuar sobre datos, también se pueden construir funciones más complejas utilizando las reglas anteriormente descritas.

## 5. Q con ejemplos

A continuación desarrollamos los ejemplos que aparecen en el documento *Q by example* y que como ya hemos indicado se puede encontrar en la página web de Kx Systems.

### 5.1 Aritmética simple

```
q)2+2           /comentario es ' /'
```

4

q)2-3                    /números negativos

-1

q)2\*3+4                /evaluación de derecha a izquierda

14

q)(2\*3)+4              /cambio de orden mediante paréntesis

10

q)3%4                   /división representada por '%'

0.75

q){x\*x}4                /elevar al cuadrado

16

q)sqrt 4                /raíz cuadrada

2.0

q)reciprocal 4    / 1/x inverso

0.25



## 5.2 Operaciones con listas

```
q)2*1 2 3      /lista tipo numérico con separación por espacios
2 4 6

q)1 2 3%2 4 6  /operaciones lista a lista, mismo espacio
0.5 0.5 0.5

q)count 1 2 3  /tamaño del vector
3

q)3#1           /genera secuencia de números
1 1 1

q)5#1 2         /genera una lista a partir de elementos dados
1 2 1 2 1
```

## 5.3 Elementos de una lista

```
q)first 1 2 3   /primer elemento
1

q)last 1 2 3    /último elemento
3

q)1_1 2 3       /elimina el primer elemento
2 3

q)-1_1 2 3      /elimina el último elemento
1 2

q)reverse 1 2 3 /invierte
3 2 1
```

## 5.4 Indexado y clasificación

```
q)1 2 3@1      /el indexado está basado en el cero
2

q)1 2 3@1 0    /el indexado puede generar también vectores
2 1

q)til 3        /genera secuencia de enteros
0 1 2

q)2 4 6?4      /índice de un elemento dado
1

q)iasc 2 1 6   / índice ordenado
1 0 2

q)asc 2 1 6    /vector ordenado
`s#1 2 6
```

## 5.5 Agregación de listas

```
q)1 2 3,10 20  /une listas
1 2 3 10 20

q)1+2+3        /suma de elementos
6

q)sum 1 2 3    /inserta '+' entre los elementos
6

q)sums 1 2 3   /sumas consecutivas de elementos
1 3 6

q)1,(1+2),(1+2+3) /lo mismo que esto
1 3 6

q){1_x+prev x}til 5  /sum running pairs
1 3 5 7
```

```
q)sum each{(2*til ceiling .5*count x)_x}1 2 3 4 5 /non-  
intersecting pairs
```

```
3 7 5
```

```
q)(1 2;3 4 6;7 6) /lista
```

```
(1 2;3 4 6;7 6)
```

```
q)first(3 4 6;7 6) /primer elemento de la lista
```

```
3 4 6
```

## 5.6 Combinación de funciones

```
q){x+x*x}4 /a + a^2
```

```
20
```

```
q)(sqrt;{x*x})@\:4 /[sqrt(a), a^2]
```

```
(2f;16)
```

```
q){x*x}sum 2 3 /(a +b)^2
```

```
25
```

```
q)sum{x*x}2 3 /a^2 + b^2
```

```
13
```

```
q){sum(x*x),2*/x}2 3 /(a + b)^2 = a^2 + b^2 + 2ab
```

```
25
```

```
q)sqrt sum{x*x}3 4 /sqrt(a^2 + b^2)
```

```
5f
```

## 5.7 Funciones definidas por el usuario y argumentos

```
q)d1:- /proyección diádica
```

```
q)d2:{x-y} /función diádica explícita
```

```
q)m1:neg /proyección monádica
```

```
q)m2:0- /proyección monádica
```

```

q)m3:{neg x}      /función monádica explícita

q)(m1;m2;m3)@\:4    / "monads"

-4 -4 -4

q)(d1;d2).\:3 4      / "dyads"

-1 -1

```

## 6.8 Exponenciación y logaritmo

```

q)(e;2*e;e*e:exp 1)  /e, 2e, e elevado al cuadrado

2.718282 5.436564 7.389056

q)exp 2              /exponencial, e^2

7.389056

q)2 xexp 16          /exponencial en base 2, 2^16

65536.0

q)log exp 2          /logaritmo, ln e^2

2.0

q)2 xlog 65536       /logaritmo en base 2, log2 65536

16.0

```

## 6.9 Trigonometría

```

q)a:(pi;2*pi;pi*pi:acos -1) /pi, 2 pi, pi al cuadrado

3.141593 6.283185 9.869604

q)cos pi            /coseno de pi

-1.0

q)(t:sum{x*x}@(cos;sin)@\:)pi  /teorema de la trigonometría

.0

```

## 5.10 Matrices

```
q)1 2 3*/:1 2 3      /tabla de multiplicaciones
(1 2 3;2 4 6;3 6 9)

q){x=/:x}@key 3      /matriz identidad
(100b;010b;001b)

q)2 3#key 6          /generar matriz
(0 1 2;3 4 5)

q)2 2#0 1 1 1        /transforma un vector dado a matriz
(0 1;1 1)
```

## 5.11 Transformaciones estructurales

```
q)raze/[N:0 3_/:2 6#key 12]      /ravel: lista de atomos
0 1 2 3 4 5 6 7 8 9 10 11

q)raze each N                  /ravel cada submatriz
(0 1 2 3 4 5;6 7 8 9 10 11)

q)M:3 3#"ABC123!@#" /matriz de caracteres

(::::flip;reverse;reverse each;1 rotate)@\:M

(("ABC";"123";"!@#");("A1!";"B2@";"C3#");("!"@#";"123";"ABC");("CBA"
;"321";"#@!");("123";"!@#";"ABC"))

q)M ./:/:f value group sum each f:n cross n:til 3      /secondary
diagonals

(enlist"A";"B1";"C2!";"3@";enlist"#")

q)M ./:a,'a:key count M /main diagonal

"A2#"
```

## 5.12 Selección

```
q)N:((0 1 2;3 4 5);(6 7 8;9 10 11))

q)((N 1) 1) 1 /selección repetitiva de elementos de una lista
10

q)3@[;1]/N      /aplica tres veces la selección
10

q)N[1;1;1]      / Selección transversal
10

q)N . 1 1 1      /también selección transversal
10
```

## 5.13 Factorial y Coeficientes Binomiales

```
q)each[f:{$[x<0;0;prd 1.+til x]]1+til 5 /factorial
1 2 6 24 120.0

q)prds 1+til 5                                /running product
1 2 6 24 120

q)(b:{til[x]{$[x<y;0;floor f[x]%f[y]*f x-y]}\:/:til x})5
/coeficientes binomiales

(1 1 1 1 1;0 1 2 3 4;0 0 1 3 6;0 0 0 1 4;0 0 0 0 1)

q)1_{sum b[x]./:flip(til x;reverse key x)}each til 16 /fibonacci:
sum of second diagonal of binomial matrix
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

## 5.14 Producto Escalar

```
q)1 2 3 wsum 1 2 3      /product escalar wsum=+/* (optimizado)
14f

q)1 2 3.$1 2 3.        /también
```

```

14f
q)M:(0 1.;1 1.) /asignación
q)M$M /matriz al cuadrado optimizado)
(1 1.;1 2.)
q)15$[M]/M /matrix to the power of 15, also fibonacci
(610 987.;987 1597.)
q)(14$[M]\M)[;0;1]
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610f

```

## 5.15 Probabilidad y azar

```

q)A:5?1.;A /5 cinco números aleatorios 0..1
0.03505812 0.7834427 0.7999031 0.9046515 0.2232866
q)B:10?2;B /lanzamiento de monedas
1 1 1 0 1 0 1 1 0 0
q)B1:10?0b;B1 /con úneros booleanos
11110010101b
q)C:-3?3;C /deal 3 unique cards out of 3
1 0 2
q)(min;max)@\ :A /minimo y máximo de una lista
0.03505812 0.9046515
q)B?0 /first zero
3
q)avg C~/ :1_10000{-3?3}\() /método de monte carlo
0.1643836
q)reciprocal f 3 /exact probability of 3 cards in given
order
0.1666667

```

## 5.16 Elementos únicos

```
q)D:distinct S:"mississippi" /elementos distintos
"misp"

q)K:D?S;K                               /find (?) indexes
0 1 2 2 1 2 2 1 3 3 1

q)S value group K                         /group by key
(enlist "m"; "iiii"; "ssss"; "pp")

q)count each group S                     /frecuencias
"misp"!1 4 4 2

q)I:(key count S)in first each group S;I /sieve of nub
11100000100b                           /where D is in S

q)S where I                              /filter by sieve to get D
"misp"

q)sum D=/:S                              /where items of D are in S
1 4 4 2
```

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*

\*

<http://sites.google.com/site/jforscience/>

journalofj@hotmail.com