Estructuras de Datos en Maple

José Luis Torres Rodríguez*

Febrero 2011

Muchos de los datos usados en Maple son manejados por medio de varias estructuras soportadas por este sistema. Es importante conocer estas estructuras, ya que son utilizadas por muchas de las instrucciones proporcionadas, así como por el lenguaje de programación soportado. Por ejemplo, para graficar dos funciones de una variable en un mismo despliegue, utilizamos la instrucción **plot** de la siguiente manera:

plot({función1, función2}, rango);

Como puede observarse, las funciones a graficar deben colocarse en la forma **"{función1, función2}"**, es decir, como un conjunto. En muchas otras instrucciones los datos deben ser introducidos utilizando una estructura; además, en muchos casos los resultados tambien son proporcionados al usuario con una de estas estructuras.

1. Obtención del tipo de una expresión

Todos los elementos utilizados por Maple son clasificados en diferentes tipos; para poder manipularlos correctamente es útil saber a cual de estos pertenecen. La instruccion que nos muestra esta información es **whattype**. Veamos algunos ejemplos:

```
> whattype(3*x);
```

*

```
> whattype([3*x]);
```

list

```
> whattype({3*x});
```

set

La primera expresión es reconocida por Maple como un producto (**"*"**), la segunda como una lista (**"list"**) y la tercera como un conjunto (**"set"**).

Maple puede reconocer el tipo de cualquier expresión, por ejemplo números o expresiones algebraicas. Los números se clasifican segun sean enteros (integer), fracciones o decimales (float). Si son expresiones algebraicas, Maple las clasifica segun la operacion algebraica principal. A continuacion algunos ejemplos más:

1.1. Números

Por ejemplo un entero, una fraccion, un irracional y un numero decimal:

```
> whattype(5!);
```

integer

*Coordinación de Cómputo, Facultad de Ciencias, UNAM

```
> whattype(12/31);
```

fraction

```
> whattype(sqrt(Pi));
```

^

```
> whattype(1.2783);
```

float

Nótese que el tipo del irracional **sqrt(Pi)** es **"^"** (exponenciación), puesto que: **sqrt(Pi)=Pi^(1/2)**.
Verifiquemos el tipo de los siguientes datos:

```
> whattype(2^(1/2));
```

^

```
> whattype(3*4);
```

integer

```
> whattype(3*4 + 1);
```

integer

Tal vez esperariamos que **"2^(1/2)"** fuera reconocido como ***"float"***, sin embargo, en este caso Maple la reconoce como si se tratara de una expresión algebraica. Para determinar el tipo de estas expresiones, Maple toma en cuenta la precedencia de los operadores que aparecen en ellas.

1.2. Expresiones algebraicas

En expresiones simples, el tipo viene dado por la operación. En este caso, suma y resta, multiplicación y división, potencia y raiz, son consideradas como una sola operación, respectivamente:

```
> whattype(x + 3);
```

+

```
> whattype(3/x);
```

*

```
> whattype(x^3);
```

^

```
> whattype(x^(1/3));
```

^

En expresiones que involucran varias operaciones, el tipo viene dado por la operación que describe globalmente a la expresión. Por ejemplo:

```
> (3*x^2 + 3*y - 3*z)/(1 + x*y*z);
```

$$\frac{3\,x^2 + 3\,y - 3\,z}{1 + x\,y\,z}$$

```
> whattype(%);
```

*

Ésta última es reconocida como un producto (aunque hay sumas y potencias, estas no describen globalmente a la expresion, sino solo partes de ella).

Finalmente, determinemos el tipo de las siguientes expresiones (podemos identificar en cada una de ellas la operación que las describe):

```
> whattype(3*x^2/(1 + x*y*z) + 3*y - 3*z);
```

+

```
> whattype(3 + ((2*sin(x)^2 - sqrt(2)*cos(x))/(2*sin(x)^2 +
> sqrt(2)*cos(x))));
```

+

1.3. Relaciones de equivalencia

Si se trata de una ecuación o desigualdad, el signo de relación determina su tipo:

```
> whattype(3*x^2 - sqrt(y - 3) = log(1 + x*y));
```

=

```
> whattype(3 < x^3);
```

<

```
> whattype(4*x - 5*y >= 23*z);
```

<=

```
> whattype(4*x - 5*y <= 23*z);
```

<=

Notese que "`>=`" y "`<=`" son considerados ambos como "`<=`". De la misma forma, "`<`" y "`>`" son considerados como "`<`".

1.4. Nombres

Si tienen alguna expresión asignada, el tipo es aquel de la expresión (o número) asignado, sin importar la forma que tenga el nombre:

```
> expr_1 := (1 + 3*x*cos(y) - a)/(1 - 3*x*cos(y) - a);
```

$$expr\_1 := \frac{1 + 3\,x\cos(y) - a}{1 - 3\,x\cos(y) - a}$$

```
> whattype(expr_1);
```

*

```
> `masa del objeto` := expr_1 + sin(x)/(1 + y);
```

$$masa\ del\ objeto := \frac{1 + 3\,x\cos(y) - a}{1 - 3\,x\cos(y) - a} + \frac{\sin(x)}{1 + y}$$

```
> whattype(`masa del objeto`);
```

+

```
> phi[0] := 1 + tan(x)^2*y;
```

$$\phi_0 := 1 + \tan(x)^2\,y$$

```
> whattype(phi[0]);
```

+

Si se trata de una expresión a la cual no se le ha asignado un valor, ésta es reconocida como ***"symbol"***:

```
> whattype(masa);
```

symbol

Una excepción son las expresiones en las cuales aparecen subindices y aquellas que contienen funciones con valores indeterminados. Las primeras son reconocidas como expresiones ***"indexadas"*** y las segundas como ***"funciones"***.

```
> h[3];
```

$$h_3$$

```
> whattype(h[3]);
```

indexed

```
> whattype(sin(x*y));
```

function

```
> whattype(B(t));
```

function

1.5. Funciones

Si se definen como operador, el tipo del nombre es ***"symbol"***:

```
> Fo := (t, a) -> sqrt(1 + a*sin(t/a));
```

$$Fo := (t,\, a) \rightarrow \sqrt{1 + a \sin\left(\frac{t}{a}\right)}$$

```
> whattype(Fo);
```

symbol

Si escribimos la función con sus argumentos el tipo es el de la regla de correspondencia tomada como expresión algebraica:

```
> whattype(Fo(t, a));
```

^

Otra de las funciones útiles para verificar el tipo de un dato es **type**, su sintaxis es la siguiente:

type(expresión, tipo);

Esta función devuelve ***"true"*** si la expresión dada es del tipo especificado y ***"false"*** en otro caso. Por ejemplo:

```
> type(1 - 2*I, complex);
```

true

```
> type(1/4, fraction);
```

true

```
> type(hola, integer);
```

false

Vease la página de ayuda de type para obtener una lista completa de los tipos reconocidos.

Otra función, útil también para este tipo de operaciones es **hastype**, su sintaxis es:

hastype(expresión, tipo);

Esta función devuelve ***"true"*** si la expresión dada contiene algún elemento del tipo especificado. Por ejemplo:

```
> hastype(x^2 + 4*x, `+`);
```

true

En este caso el valor devuelto es ***"true"***, pues la expresión involucrada contiene una suma. Los tipos de datos soportados por esta función son los mismos que los de la función **type**.

2. Tipos de estructuras

Los principales tipos de estructuras soportados por Maple son: secuencias, conjuntos, listas, funciones, tablas y arreglos. En base a estos se manejan la mayor parte de los datos usados en este sistema. A continuación haremos una revisión de cada uno de ellos.

Nota: existe otro tipo de estructura conocido como **rtable**, el cual es usado por la función **Matrix** y por el paquete **LinearAlgebra** para creación y manejo de matrices y vectores; sin embargo, esta estructura no es utilizada en otros casos, por lo que no la describiremos aqui.

2.1. Secuencias de expresiones

Si tecleamos varias expresiones (cualquier combinación de los tipos anteriormente mencionados) separadas por comas, obtenemos una estructura llamada ***"secuencia de expresiones"***, que Maple reconoce con el nombre ***"exprseq"***. Por ejemplo:

```
> 1, 2, 3, sqrt(x + 1), sin(x), S[12], C(x);
```

$$1,\ 2,\ 3,\ \sqrt{x+1},\ \sin(x),\ S_{12},\ \mathrm{C}(x)$$

```
> whattype(%);
```

exprseq

Estas secuencias de expresiones pueden ser asignadas a variables:

```
> hola := 1.1, sqrt(23), 3, 7, 89, 23;
```

$$hola := 1{,}1,\ \sqrt{23},\ 3,\ 7,\ 89,\ 23$$

```
> whattype(hola);
```

exprseq

Existen varias forma de definir una secuencia; una de ellas es la del ejemplo anterior, simplemente colocamos los elementos de manera consecutiva separados por comas. Pero imaginemos el problema que representa formar una estructura de este tipo que contenga los primeros 100 números naturales, tecleando cada uno de ellos. Maple nos proporciona una forma de tratar con este problema, permitiendonos crear este tipo de estructuras por ***"definición secuencial"***, lo cual puede llevarse a cabo por medio de la instrucción **seq**.

La sintaxis de esta función es:

seq(r(i), i=rango);

Donde **r(i)** es una expresión en términos de la variable **i**, que determina de que manera se generan los elementos de la secuencia, y **rango** determina que valores tomará **i**; es decir, cuantos elementos y con que valores de **i** estarán dentro de esta secuencia. Por ejemplo, para obtener una secuencia que contenga los 100 primeros números naturales podemos utilizar la siguiente instrucción:

```
> seq(i, i=1..100);
```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
94, 95, 96, 97, 98, 99, 100

El índice de la secuencia (**i** en el ejemplo) puede ser cualquier cadena de caracteres (a la cual no le ha sido asignado previamente un dato), y puede tomar valores enteros negativos. Tambien es posible asignar un nombre a la secuencia obtenida. Por ejemplo, la secuencia de cuadrados de enteros en el rango -10..10 puede ser obtenida por:

```
> cuadr := seq(entero^2, entero=-10..10);
```

$$cuadr := 100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100$$

Es posible obtener de esta forma secuencias como las que se muestran a continuación:

- Secuencias a partir de funciones.

```
> C := x -> x + 4;
```

$$C := x \to x + 4$$

```
> sec1 := seq(C(k) + k*sin((k - 1)), k=1..10);
```

$$\text{sec1} := 5,\ 6 + 2\sin(1),\ 7 + 3\sin(2),\ 8 + 4\sin(3),\ 9 + 5\sin(4),\ 10 + 6\sin(5),\ 11 + 7\sin(6),\ 12 + 8\sin(7),\ 13 + 9\sin(8),\ 14 + 10\sin(9)$$

- Secuencias que involucran relaciones de equivalencia.

```
> sec2 := seq(C(n) + 1 = n*sin(n*x + 1), n=-3..4);
```

$$\text{sec2} := 2 = 3\sin(3\,x - 1),\ 3 = 2\sin(2\,x - 1),\ 4 = \sin(x - 1),\ 5 = 0,\ 6 = \sin(x + 1),\ 7 = 2\sin(2\,x + 1),\ 8 = 3\sin(3\,x + 1),\ 9 = 4\sin(4\,x + 1)$$

- Objetos indexados.

```
> sec3 := seq(A[K + 1] - B[K] + C(K), K=-5..5);
```

$$\text{sec3} := A_{-4} - B_{-5} - 1,\ A_{-3} - B_{-4},\ A_{-2} - B_{-3} + 1,\ A_{-1} - B_{-2} + 2,\ A_0 - B_{-1} + 3,\ A_1 - B_0 + 4,\ A_2 - B_1 + 5,\ A_3 - B_2 + 6,\ A_4 - B_3 + 7,\ A_5 - B_4 + 8,\ A_6 - B_5 + 9$$

- Secuencias anidadas: sec(sec()).

```
> seq(alpha*m^2 + beta*n^2, m=1..3);
```

$$\alpha + \beta\,n^2,\ 4\,\alpha + \beta\,n^2,\ 9\,\alpha + \beta\,n^2$$

```
> seq(%, n=1..3);
```

$$\alpha + \beta,\ 4\,\alpha + \beta,\ 9\,\alpha + \beta,\ \alpha + 4\,\beta,\ 4\,\alpha + 4\,\beta,\ 9\,\alpha + 4\,\beta,\ \alpha + 9\,\beta,\ 4\,\alpha + 9\,\beta,\ 9\,\alpha + 9\,\beta$$

Esta última podemos anidarla en una misma expresión:

```
> seq(seq(alpha*m^2 + beta*n^2, m=1..3), n=1..3);
```

$$\alpha + \beta,\ 4\,\alpha + \beta,\ 9\,\alpha + \beta,\ \alpha + 4\,\beta,\ 4\,\alpha + 4\,\beta,\ 9\,\alpha + 4\,\beta,\ \alpha + 9\,\beta,\ 4\,\alpha + 9\,\beta,\ 9\,\alpha + 9\,\beta$$

```
> seq(seq(P[k, l], k=1..3), l=1..4);
```

$$P_{1,1},\ P_{2,1},\ P_{3,1},\ P_{1,2},\ P_{2,2},\ P_{3,2},\ P_{1,3},\ P_{2,3},\ P_{3,3},\ P_{1,4},\ P_{2,4},\ P_{3,4}$$

- Particiones de intervalos en los reales. Por ejemplo, el intervalo x=0..1 en diez partes iguales.

```
> seq(i/10, i=0..10);
```

$$0,\ \frac{1}{10},\ \frac{1}{5},\ \frac{3}{10},\ \frac{2}{5},\ \frac{1}{2},\ \frac{3}{5},\ \frac{7}{10},\ \frac{4}{5},\ \frac{9}{10},\ 1$$

2.1.1. Otros ejemplos

Mediante una secuencia crearemos las siguientes particiones de intervalos de la recta de los reales. Por razones de espacio no mostraremos todas las salidas, pues algunas de ellas son bastante extensas.

- El intervalo de [0..1] en 20 partes iguales:

```
> seq(i/20, i=0..20);
```

$$0, \frac{1}{20}, \frac{1}{10}, \frac{3}{20}, \frac{1}{5}, \frac{1}{4}, \frac{3}{10}, \frac{7}{20}, \frac{2}{5}, \frac{9}{20}, \frac{1}{2}, \frac{11}{20}, \frac{3}{5}, \frac{13}{20}, \frac{7}{10}, \frac{3}{4}, \frac{4}{5}, \frac{17}{20}, \frac{9}{10}, \frac{19}{20}, 1$$

- El intervalo [-7..5.33] en 59 partes iguales:

```
> seq(-7 + (5.33 - (-7))*i/59, i=0..59);
```

−7., −6,791016949, −6,582033898, −6,373050848, −6,164067797, −5,955084746, −5,746101695, −5,537118644, −5,328135594, −5,119152543, −4,910169492, −4,701186441, −4,492203390, −4,283220340, −4,074237289, −3,865254238, −3,656271187, −3,447288136, −3,238305086, −3,029322035, −2,820338984, −2,611355933, −2,402372882, −2,193389832, −1,984406781, −1,775423730, −1,566440679, −1,357457628, −1,148474578, −0,939491527, −0,730508476, −0,521525425, −0,312542374, −0,103559324, 0,105423727, 0,314406778, 0,523389829, 0,732372880, 0,941355930, 1,150338981, 1,359322032, 1,568305083, 1,777288134, 1,986271184, 2,195254235, 2,404237286, 2,613220337, 2,822203388, 3,03118644, 3,24016949, 3,44915254, 3,65813559, 3,86711864, 4,07610169, 4,28508474, 4,49406779, 4,70305084, 4,91203390, 5,12101695, 5,33000000

- Definir quince angulos iguales en un circulo:

```
> seq(2*Pi*i/15, i=0..15);
```

$$0, \frac{2\,\pi}{15}, \frac{4\,\pi}{15}, \frac{2\,\pi}{5}, \frac{8\,\pi}{15}, \frac{2\,\pi}{3}, \frac{4\,\pi}{5}, \frac{14\,\pi}{15}, \frac{16\,\pi}{15}, \frac{6\,\pi}{5}, \frac{4\,\pi}{3}, \frac{22\,\pi}{15}, \frac{8\,\pi}{5}, \frac{26\,\pi}{15}, \frac{28\,\pi}{15}, 2\,\pi$$

Ahora veamos algunos ejemplos de particiones en el plano:

- Una malla de 10X20 en los rangos: x=-2..2, y=0..3:

```
> seq(seq([-2 + 4*n/9, 3*m/19], m=0..19), n=0..9);
```

$$[-2, 0], [-2, \frac{3}{19}], [-2, \frac{6}{19}], [-2, \frac{9}{19}], [-2, \frac{12}{19}], [-2, \frac{15}{19}], [-2, \frac{18}{19}], [-2, \frac{21}{19}], [-2, \frac{24}{19}], [-2, \frac{27}{19}],$$
$$[-2, \frac{30}{19}], [-2, \frac{33}{19}], [-2, \frac{36}{19}], [-2, \frac{39}{19}], [-2, \frac{42}{19}], [-2, \frac{45}{19}], [-2, \frac{48}{19}], [-2, \frac{51}{19}], [-2, \frac{54}{19}],$$
$$[-2, 3], [\frac{-14}{9}, 0], [\frac{-14}{9}, \frac{3}{19}], [\frac{-14}{9}, \frac{6}{19}], [\frac{-14}{9}, \frac{9}{19}], [\frac{-14}{9}, \frac{12}{19}], [\frac{-14}{9}, \frac{15}{19}], [\frac{-14}{9}, \frac{18}{19}],$$
$$[\frac{-14}{9}, \frac{21}{19}], [\frac{-14}{9}, \frac{24}{19}], [\frac{-14}{9}, \frac{27}{19}], [\frac{-14}{9}, \frac{30}{19}], [\frac{-14}{9}, \frac{33}{19}], [\frac{-14}{9}, \frac{36}{19}], [\frac{-14}{9}, \frac{39}{19}], [\frac{-14}{9}, \frac{42}{19}],$$
$$[\frac{-14}{9}, \frac{45}{19}], [\frac{-14}{9}, \frac{48}{19}], [\frac{-14}{9}, \frac{51}{19}], [\frac{-14}{9}, \frac{54}{19}], [\frac{-14}{9}, 3], [\frac{-10}{9}, 0], [\frac{-10}{9}, \frac{3}{19}], [\frac{-10}{9}, \frac{6}{19}],$$
$$[\frac{-10}{9}, \frac{9}{19}], [\frac{-10}{9}, \frac{12}{19}], [\frac{-10}{9}, \frac{15}{19}], [\frac{-10}{9}, \frac{18}{19}], [\frac{-10}{9}, \frac{21}{19}], [\frac{-10}{9}, \frac{24}{19}], [\frac{-10}{9}, \frac{27}{19}], [\frac{-10}{9}, \frac{30}{19}],$$
$$[\frac{-10}{9}, \frac{33}{19}], [\frac{-10}{9}, \frac{36}{19}], [\frac{-10}{9}, \frac{39}{19}], [\frac{-10}{9}, \frac{42}{19}], [\frac{-10}{9}, \frac{45}{19}], [\frac{-10}{9}, \frac{48}{19}], [\frac{-10}{9}, \frac{51}{19}], [\frac{-10}{9}, \frac{54}{19}],$$
$$[\frac{-10}{9}, 3], [\frac{-2}{3}, 0], [\frac{-2}{3}, \frac{3}{19}], [\frac{-2}{3}, \frac{6}{19}], [\frac{-2}{3}, \frac{9}{19}], [\frac{-2}{3}, \frac{12}{19}], [\frac{-2}{3}, \frac{15}{19}], [\frac{-2}{3}, \frac{18}{19}], [\frac{-2}{3}, \frac{21}{19}],$$
$$[\frac{-2}{3}, \frac{24}{19}], [\frac{-2}{3}, \frac{27}{19}], [\frac{-2}{3}, \frac{30}{19}], [\frac{-2}{3}, \frac{33}{19}], [\frac{-2}{3}, \frac{36}{19}], [\frac{-2}{3}, \frac{39}{19}], [\frac{-2}{3}, \frac{42}{19}], [\frac{-2}{3}, \frac{45}{19}], [\frac{-2}{3}, \frac{48}{19}],$$
$$[\frac{-2}{3}, \frac{51}{19}], [\frac{-2}{3}, \frac{54}{19}], [\frac{-2}{3}, 3], [\frac{-2}{9}, 0], [\frac{-2}{9}, \frac{3}{19}], [\frac{-2}{9}, \frac{6}{19}], [\frac{-2}{9}, \frac{9}{19}], [\frac{-2}{9}, \frac{12}{19}], [\frac{-2}{9}, \frac{15}{19}],$$
$$[\frac{-2}{9}, \frac{18}{19}], [\frac{-2}{9}, \frac{21}{19}], [\frac{-2}{9}, \frac{24}{19}], [\frac{-2}{9}, \frac{27}{19}], [\frac{-2}{9}, \frac{30}{19}], [\frac{-2}{9}, \frac{33}{19}], [\frac{-2}{9}, \frac{36}{19}], [\frac{-2}{9}, \frac{39}{19}], [\frac{-2}{9}, \frac{42}{19}],$$
$$[\frac{-2}{9}, \frac{45}{19}], [\frac{-2}{9}, \frac{48}{19}], [\frac{-2}{9}, \frac{51}{19}], [\frac{-2}{9}, \frac{54}{19}], [\frac{-2}{9}, 3], [\frac{2}{9}, 0], [\frac{2}{9}, \frac{3}{19}], [\frac{2}{9}, \frac{6}{19}], [\frac{2}{9}, \frac{9}{19}], [\frac{2}{9}, \frac{12}{19}],$$
$$[\frac{2}{9}, \frac{15}{19}], [\frac{2}{9}, \frac{18}{19}], [\frac{2}{9}, \frac{21}{19}], [\frac{2}{9}, \frac{24}{19}], [\frac{2}{9}, \frac{27}{19}], [\frac{2}{9}, \frac{30}{19}], [\frac{2}{9}, \frac{33}{19}], [\frac{2}{9}, \frac{36}{19}], [\frac{2}{9}, \frac{39}{19}], [\frac{2}{9}, \frac{42}{19}],$$
$$[\frac{2}{9}, \frac{45}{19}], [\frac{2}{9}, \frac{48}{19}], [\frac{2}{9}, \frac{51}{19}], [\frac{2}{9}, \frac{54}{19}], [\frac{2}{9}, 3], [\frac{2}{3}, 0], [\frac{2}{3}, \frac{3}{19}], [\frac{2}{3}, \frac{6}{19}], [\frac{2}{3}, \frac{9}{19}], [\frac{2}{3}, \frac{12}{19}],$$
$$[\frac{2}{3}, \frac{15}{19}], [\frac{2}{3}, \frac{18}{19}], [\frac{2}{3}, \frac{21}{19}], [\frac{2}{3}, \frac{24}{19}], [\frac{2}{3}, \frac{27}{19}], [\frac{2}{3}, \frac{30}{19}], [\frac{2}{3}, \frac{33}{19}], [\frac{2}{3}, \frac{36}{19}], [\frac{2}{3}, \frac{39}{19}], [\frac{2}{3}, \frac{42}{19}],$$
$$[\frac{2}{3}, \frac{45}{19}], [\frac{2}{3}, \frac{48}{19}], [\frac{2}{3}, \frac{51}{19}], [\frac{2}{3}, \frac{54}{19}], [\frac{2}{3}, 3], [\frac{10}{9}, 0], [\frac{10}{9}, \frac{3}{19}], [\frac{10}{9}, \frac{6}{19}], [\frac{10}{9}, \frac{9}{19}], [\frac{10}{9}, \frac{12}{19}],$$
$$[\frac{10}{9}, \frac{15}{19}], [\frac{10}{9}, \frac{18}{19}], [\frac{10}{9}, \frac{21}{19}], [\frac{10}{9}, \frac{24}{19}], [\frac{10}{9}, \frac{27}{19}], [\frac{10}{9}, \frac{30}{19}], [\frac{10}{9}, \frac{33}{19}], [\frac{10}{9}, \frac{36}{19}], [\frac{10}{9}, \frac{39}{19}],$$
$$[\frac{10}{9}, \frac{42}{19}], [\frac{10}{9}, \frac{45}{19}], [\frac{10}{9}, \frac{48}{19}], [\frac{10}{9}, \frac{51}{19}], [\frac{10}{9}, \frac{54}{19}], [\frac{10}{9}, 3], [\frac{14}{9}, 0], [\frac{14}{9}, \frac{3}{19}], [\frac{14}{9}, \frac{6}{19}],$$
$$[\frac{14}{9}, \frac{9}{19}], [\frac{14}{9}, \frac{12}{19}], [\frac{14}{9}, \frac{15}{19}], [\frac{14}{9}, \frac{18}{19}], [\frac{14}{9}, \frac{21}{19}], [\frac{14}{9}, \frac{24}{19}], [\frac{14}{9}, \frac{27}{19}], [\frac{14}{9}, \frac{30}{19}], [\frac{14}{9}, \frac{33}{19}],$$
$$[\frac{14}{9}, \frac{36}{19}], [\frac{14}{9}, \frac{39}{19}], [\frac{14}{9}, \frac{42}{19}], [\frac{14}{9}, \frac{45}{19}], [\frac{14}{9}, \frac{48}{19}], [\frac{14}{9}, \frac{51}{19}], [\frac{14}{9}, \frac{54}{19}], [\frac{14}{9}, 3], [2, 0],$$
$$[2, \frac{3}{19}], [2, \frac{6}{19}], [2, \frac{9}{19}], [2, \frac{12}{19}], [2, \frac{15}{19}], [2, \frac{18}{19}], [2, \frac{21}{19}], [2, \frac{24}{19}], [2, \frac{27}{19}], [2, \frac{30}{19}],$$
$$[2, \frac{33}{19}], [2, \frac{36}{19}], [2, \frac{39}{19}], [2, \frac{42}{19}], [2, \frac{45}{19}], [2, \frac{48}{19}], [2, \frac{51}{19}], [2, \frac{54}{19}], [2, 3]$$

Podemos verificar que la cantidad de elementos generados por esta última instrucción, utilizando la instrucción **nops**, la cual nos devuelve el número de elementos que contiene una lista.

Para que considere a esta secuencia como lista basta con encerrarla entre corchetes.

```
> nops([%]);
```

200

- Una malla 17X13 para los intervalos x=-Pi..Pi, y=0..2*Pi:

```
> seq(seq([-Pi + 2*Pi*n/9, 2*Pi*m/19], m=0..12), n=0..16):
```

Por razones de espacio no mostramos los elementos generados; sin embargo, podemos comprobar el número de elementos generadosla cantidad de éstos:

```
> nops([%]);
```

221

- Finalmente, veamos el siguiente ejemplo:

Considerese la secuencia **S := seq(A[k] + cos(1 + B[k + 1]), k=-3..3)** de 7 elementos.

¿Cómo podemos formar la secuencia de **A[k] + cos(1 + B[k + 1])**, de tal manera que **k** tome siete valores diferentes entre -3/2 y 3/2?

```
> S := seq(A[k/2] + cos(1 + B[k/2 + 1]), k=-3..3);
```

$S := A_{-3/2} + \cos(1 + B_{-1/2}),\ A_{-1} + \cos(1 + B_0),\ A_{-1/2} + \cos(1 + B_{1/2}),\ A_0 + \cos(1 + B_1),$
$A_{1/2} + \cos(1 + B_{3/2}),\ A_1 + \cos(1 + B_2),\ A_{3/2} + \cos(1 + B_{5/2})$

2.1.2. Acceso a los elementos de una secuencia

Consideremos las siguientes secuencias:

```
> S_a := seq(cos(x + i/5), i=0..5);
```

$$S\_a := \cos(x), \cos\left(x + \frac{1}{5}\right), \cos\left(x + \frac{2}{5}\right), \cos\left(x + \frac{3}{5}\right), \cos\left(x + \frac{4}{5}\right), \cos(x + 1)$$

```
> S_b := seq(y + i/6, i=0..6);
```

$$S\_b := y, y + \frac{1}{6}, y + \frac{1}{3}, y + \frac{1}{2}, y + \frac{2}{3}, y + \frac{5}{6}, 1 + y$$

Si queremos extraer el tercer elemento de **S_a** , usamos subíndices de la siguiente forma:

```
> S_a[3];
```

$$\cos\left(x + \frac{2}{5}\right)$$

Es posible formar subsecuencias a partir de secuencias ya definidas. Para esto hay dos formas equivalentes; por ejemplo, podemos hacerlo utilizando la instrucción **seq** de la siguiente manera:

```
> seq(S_a[j], j=2..5);
```

$$\cos\left(x + \frac{1}{5}\right), \cos\left(x + \frac{2}{5}\right), \cos\left(x + \frac{3}{5}\right), \cos\left(x + \frac{4}{5}\right)$$

Con esto extraemos los elementos del segundo al quinto de **S_a**. Otra forma es la siguiente:

```
> S_a[2..5];
```

$$\cos\left(x + \frac{1}{5}\right), \cos\left(x + \frac{2}{5}\right), \cos\left(x + \frac{3}{5}\right), \cos\left(x + \frac{4}{5}\right)$$

También podemos definir secuencias nuevas en base a las secuencia previamente definidas. Por ejemplo, la secuencia de cosenos de cada elemento de **S_b** está dada por:

```
> S_c := seq(cos(S_b[i]), i=1..7);
```

$$S\_c := \cos(y), \cos\left(y+\frac{1}{6}\right), \cos\left(y+\frac{1}{3}\right), \cos\left(y+\frac{1}{2}\right), \cos\left(y+\frac{2}{3}\right), \cos\left(y+\frac{5}{6}\right), \cos(1+y)$$

Nota: al definir **S_c**, el indice **i** no puede tomar valores negativos ni cero, pues denota un orden de los elementos de **S_b** (no hay elemento ***"cero-esimo"***, ni ***"menos cinco"***). También, el maximo valor del índice no debe exceder el numero de elementos de **S_b** (si tiene 7 elementos, no podemos invocar al octavo elemento).

2.2. Listas y conjuntos

Las listas y los conjuntos están formados por expresiones y secuencias de expresiones. Existe una diferencia muy importante entre las listas y los conjuntos. En el caso de las listas el orden de los elementos es importante; en cambio en los conjuntos el orden es irrelevante*(* al igual que en los conjuntos matemáticos).

Por otro lado, los elementos de listas y conjuntos se pueden definir como en las secuencias, proporcionando cada uno de los elementos separados por coma o bien por definicion secuencial. La sintaxis para crear una de estas estructuras se muestra a continuación.

Para definir una lista:

[e1, e2, e3, ...] , donde **"e1, e2, e3, ..."** son los elementos.

O bien:

[seq(...)] , utilizando la definición secuencial.

El caso de los conjuntos es análogo, pero los elementos deben estar delimitados por llaves (**"{ }"**):

{e1, e2, e3, ...} , donde **"e1, e2, e3, ..."** son los elementos.

O bien,

{seq(...)} , utilizando la definición secuencial.

Maple considera a las listas, los conjuntos y los elementos que los forman como estructuras diferentes. Por ejemplo, definamos la siguiente secuencia:

```
> sq := seq(i/9, i=0..9);
```

$$sq := 0, \frac{1}{9}, \frac{2}{9}, \frac{1}{3}, \frac{4}{9}, \frac{5}{9}, \frac{2}{3}, \frac{7}{9}, \frac{8}{9}, 1$$

Veamos el tipo de **sq** y los tipos de la lista y el conjunto formados por sus elementos:

```
> whattype(sq);
```

$$exprseq$$

```
> listasq := [sq];
```

$$listasq := \left[0, \frac{1}{9}, \frac{2}{9}, \frac{1}{3}, \frac{4}{9}, \frac{5}{9}, \frac{2}{3}, \frac{7}{9}, \frac{8}{9}, 1\right]$$

```
> whattype(listasq);
```

$$list$$

```
> conjsq := {sq};
```

$$conjsq := \left\{0, 1, \frac{5}{9}, \frac{1}{3}, \frac{2}{3}, \frac{4}{9}, \frac{2}{9}, \frac{8}{9}, \frac{1}{9}, \frac{7}{9}\right\}$$

```
> whattype({conjsq});
```

$$set$$

Nótese que una forma de crear una lista o un conjunto es encerrando entre corchetes o llaves (respectivamente) los elementos de una secuencia.

Para ilustrar la diferencia entre listas y conjuntos, examinemos el ejemplo siguiente.

Definimos la secuencia **elems**:

```
> elems := a, a, a, a, 1+a, sin(a), sin(1 + a);
```

$$elems := a,\ a,\ a,\ a,\ 1 + a,\ \sin(a),\ \sin(1 + a)$$

A continuación utilizamos esta secuencia para crear una lista o un conjunto.

```
> Lista := [elems];
```

$$Lista := [a,\ a,\ a,\ a,\ 1 + a,\ \sin(a),\ \sin(1 + a)]$$

```
> Conjunto := {elems};
```

$$Conjunto := \{a,\ 1 + a,\ \sin(a),\ \sin(1 + a)\}$$

Como puede verse, los componentes de **elems** son colocados integramente en la lista; en cambio en el conjunto se eliminan los duplicados y además estos elementos no necesariamente aparecen en el mismo orden.

Por otro lado, las listas y los conjuntos pueden ser manipulados de la misma forma que las secuencias:

```
> Lista;
```

$$[a,\ a,\ a,\ a,\ 1 + a,\ \sin(a),\ \sin(1 + a)]$$

```
> Lista[2];
```

$$a$$

```
> Lista[2..5];
```

$$[a,\ a,\ a,\ 1 + a]$$

```
> Conjunto;
```

$$\{a,\ 1 + a,\ \sin(a),\ \sin(1 + a)\}$$

```
> Conjunto[2];
```

$$1 + a$$

```
> Conjunto[1..3];
```

$$\{a,\ 1 + a,\ \sin(a)\}$$

Nota: aunque el orden no es importante en los conjuntos, al hacer referencia a un elemento, por ejemplo **Conjunto[2]**, el elemento que devuelve Maple es, en este caso, el segundo dependiendo del orden que tenia la salida cuando éste fue definido.

Para invocar a todos los elementos de una lista o conjunto usamos la instrucción **op** y para obtener el número de elementos usamos **nops**, como se muestra a continuación:

```
> Lista;
```

$$[a,\ a,\ a,\ a,\ 1 + a,\ \sin(a),\ \sin(1 + a)]$$

```
> op(Lista);
```

$$a,\ a,\ a,\ a,\ 1 + a,\ \sin(a),\ \sin(1 + a)$$

```
> nops(Lista);
```

$$7$$

```
> Conjunto;
```

$$\{a,\ 1+a,\ \sin(a),\ \sin(1+a)\}$$

```
> op(Conjunto);
```

$$a,\ 1+a,\ \sin(a),\ \sin(1+a)$$

```
> nops(Conjunto);
```

$$4$$

Por ejemplo, para formar un conjunto con los elementos de una lista podemos proceder de la siguiente forma:

```
> Lis2 := [a, c, d, g, r, t];
```

$$Lis2 := [a,\ c,\ d,\ g,\ r,\ t]$$

```
> Conj2 := {op(Lis2)};
```

$$Conj2 := \{g,\ a,\ r,\ c,\ t,\ d\}$$

La instruccion **op** también permite invocar uno o más elementos, de la siguiente forma:

```
> Lista;
```

$$[a,\ a,\ a,\ a,\ 1+a,\ \sin(a),\ \sin(1+a)]$$

```
> op(2, Lista);
```

$$a$$

```
> op(2..6, Lista);
```

$$a,\ a,\ a,\ 1+a,\ \sin(a)$$

```
> Conjunto;
```

$$\{a,\ 1+a,\ \sin(a),\ \sin(1+a)\}$$

```
> op(4, Conjunto);
```

$$\sin(1+a)$$

```
> op(2..4, Conjunto);
```

$$1+a,\ \sin(a),\ \sin(1+a)$$

Nota: las instrucciones **op** y **nops** no actuan en secuencias, solo en listas y conjuntos.

La instrucción **ops** es útil para construir listas o conjuntos nuevos, a partir de listas o conjuntos ya definidos. La instruccion **nops** también es muy útil, ya que al hacer uso de ella evitamos tener que contabilizar los elementos de la lista (o el conjunto) ya existente. Por ejemplo, a partir de los elementos de **Lista**, podemos crear facilmente otra lista nueva, aplicando la función **cos** a cada elemento:

Primero construimos los elementos de la nueva lista:

```
> elems_nuevos := seq(cos(Lista[j]), j=1..nops(Lista));
```

$$elems\_nuevos := \cos(a),\ \cos(a),\ \cos(a),\ \cos(a),\ \cos(1+a),\ \cos(\sin(a)),\ \cos(\sin(1+a))$$

Nótese que esta lista está formada únicamente por elementos simbólicos.

A continuación, utilizando esta secuencia, construimos la lista nueva:

```
> Lista_nueva := [elems_nuevos];
```

$$Lista\_nueva := [\cos(a),\ \cos(a),\ \cos(a),\ \cos(a),\ \cos(1+a),\ \cos(\sin(a)),\ \cos(\sin(1+a))]$$

Obviamente, esto lo podemos hacer en un solo paso. Por ejemplo, queremos una lista formada por los elementos de **Conjunto** elevados al cubo:

```
> Otra_Lista := [seq(Conjunto[j]^3, j=1..nops(Conjunto))];
```

$$Otra\_Lista := [a^3,\ (1+a)^3,\ \sin(a)^3,\ \sin(1+a)^3]$$

2.2.1. Listas de listas, listas de conjuntos, conjuntos de listas, ...

Es posible construir listas o conjuntos cuyos elementos son a su vez listas o conjuntos. Como ejemplo, formamos la lista cuyos elementos son ***Lista_ nueva*** y **Otra_ lista**, definidos previamente.

```
> Superlista := [Lista_nueva, Otra_Lista];
```

$$Superlista := [\,[\cos(a),\ \cos(a),\ \cos(a),\ \cos(a),\ \cos(1+a),\ \cos(\sin(a)),\ \cos(\sin(1+a))\,],\ [\,a^3,\ (1+a)^3,\ \sin(a)^3,\ \sin(1+a)^3\,]\,]$$

Ahora, **Superlista** tiene solo dos elementos (las dos listas):

```
> nops(Superlista);
```

$$2$$

```
> op(Superlista);
```

$$[\,\cos(a),\ \cos(a),\ \cos(a),\ \cos(a),\ \cos(1+a),\ \cos(\sin(a)),\ \cos(\sin(1+a))\,],\ [\,a^3,\ (1+a)^3,\ \sin(a)^3,\ \sin(1+a)^3\,]$$

El primero y el segundo elementos son las listas a partir de las cuales fue generado:

```
> Superlista[1];
```

$$[\,\cos(a),\ \cos(a),\ \cos(a),\ \cos(a),\ \cos(1+a),\ \cos(\sin(a)),\ \cos(\sin(1+a))\,]$$

```
> Superlista[2];
```

$$[\,a^3,\ (1+a)^3,\ \sin(a)^3,\ \sin(1+a)^3\,]$$

Para invocar algún(os) elemento(s) de las listas componentes a partir de la lista grande tenemos que hacerlo de manera iterada. Por ejemplo, para invocar a la expresion sin(1+a)^3:

```
> Superlista[2][4];
```

$$\sin(1+a)^3$$

Es decir, el cuarto elemento del segundo componente. O bien, podemos hacerlo de las siguientes dos formas:

```
> op(4, Superlista[2]);
```

$$\sin(1+a)^3$$

```
> op(4, op(2, Superlista));
```

$$\sin(1+a)^3$$

Un ejemplo interesante de extraccion de elementos es el que se muestra a continuación:

```
> CC := [seq([i/10, f[i] = cos(i/10), g[i] = exp(-i/10)], i=0..10)];
```

$$CC := [\,[0,\ f_0 = 1,\ g_0 = 1\,],\ [\,\frac{1}{10},\ f_1 = \cos(\frac{1}{10}),\ g_1 = e^{(\frac{-1}{10})}\,],\ [\,\frac{1}{5},\ f_2 = \cos(\frac{1}{5}),\ g_2 = e^{(-1/5)}\,],$$
$$[\,\frac{3}{10},\ f_3 = \cos(\frac{3}{10}),\ g_3 = e^{(\frac{-3}{10})}\,],\ [\,\frac{2}{5},\ f_4 = \cos(\frac{2}{5}),\ g_4 = e^{(-2/5)}\,],\ [\,\frac{1}{2},\ f_5 = \cos(\frac{1}{2}),\ g_5 = e^{(-1/2)}\,],$$
$$[\,\frac{3}{5},\ f_6 = \cos(\frac{3}{5}),\ g_6 = e^{(-3/5)}\,],\ [\,\frac{7}{10},\ f_7 = \cos(\frac{7}{10}),\ g_7 = e^{(\frac{-7}{10})}\,],\ [\,\frac{4}{5},\ f_8 = \cos(\frac{4}{5}),\ g_8 = e^{(-4/5)}\,],$$
$$[\,\frac{9}{10},\ f_9 = \cos(\frac{9}{10}),\ g_9 = e^{(\frac{-9}{10})}\,],\ [\,1,\ f_{10} = \cos(1),\ g_{10} = e^{(-1)}\,]\,]$$

Para extraer la expresion ***"cos(7/10)"*** tenemos que invocar, del octavo elemento de la lista (es decir, de la sublista número 8), el lado derecho del segundo elemento:

```
> CC[8];
```

$$\left[\frac{7}{10},\ f_7 = \cos\left(\frac{7}{10}\right),\ g_7 = e^{(\frac{-7}{10})}\right]$$

```
> CC[8][2];
```

$$f_7 = \cos\left(\frac{7}{10}\right)$$

```
> rhs(CC[8][2]);
```

$$\cos\left(\frac{7}{10}\right)$$

Otra forma de hacerlo es:

```
> rhs(op(2, CC[8]));
```

$$\cos\left(\frac{7}{10}\right)$$

O bien:

```
> rhs(op(2, op(8, CC)));
```

$$\cos\left(\frac{7}{10}\right)$$

2.3. Funciones

Otra de las estructuras soportadas por Maple son las funciones, las cuales ya han sido tratadas previamente. Veamos algunos ejemplos:

```
> f1 := x -> x^2 + 4;
```

$$f1 := x \to x^2 + 4$$

```
> f2 := (a, b) -> a^b;
```

$$f2 := (a,\ b) \to a^b$$

```
> f3 := x - y^3;
```

$$f3 := x - y^3$$

Aplicamos la función **whattype** a cada una de ellas:

```
> whattype(f1(x));
```

$$+$$

```
> whattype(f2(a, b));
```

$$\hat{}$$

```
> whattype(f3);
```

$$+$$

Como habiamos visto previamente, el tipo de cada una de las funciones está dado por el tipo de la regla de correspondencia.

2.4. Tablas

Las tablas son un tipo de estructura cuyos elementos están dados por un grupo de datos, cada uno de los cuales tiene asignado un índice. La forma de crear estas estructuras es a través de la función **table**, dando los elementos como una lista de relaciones de la siguiente forma:

table([el1 = dato1, el2 = dato2, el3 = dato3, ...]);

donde **"el1, el2, el3, ..."** son los índices de los elementos y **"dato1, dato2, dato3, ..."** son los elementos asignados a cada uno de ellos. Por ejemplo, definamos la siguiente tabla:

```
> valores := table([A=2, B=4331, C=.3382, D=sqrt(Pi^2), E=hola]);
```

$$valores := \text{table}([\, A = 2,\ E = (1{,}1,\ \sqrt{23},\ 3,\ 7,\ 89,\ 23),\ \mathrm{D} = \pi,\ C = 0{,}3382,\ B = 4331\,])$$

El acceso a un elemento de una tabla se hace a traves de los índices. Por ejemplo, para acceder a los elementos indexados por **D** y **B**, usamos la siguiente instrucción:

```
> valores[D];
```

$$\pi$$

```
> valores[B];
```

$$4331$$

Los índices utilizados en una tabla pueden estar formados por cualquier expresión válida, incluso pueden incluirse expresiones con espacios, siempre que sean colocadas como cadenas. Por ejemplo:

```
> t := table([‘dato 1‘ = ‘cadena de caracteres‘, ‘dato 2‘ = 75452]);
```

$$t := \text{table}([\, dato\ 2 = 75452,\ dato\ 1 = cadena\ de\ caracteres\,])$$

Para acceder a los elementos también es necesario expresar los índices como cadenas (en este caso).

```
> t[‘dato 1‘];
```

$$cadena\ de\ caracteres$$

```
> t[‘dato 2‘];
```

$$75452$$

También pueden usarse números como índices.

```
> t2 := table([1 = 67832, 2 = ‘cadena x‘, 5 = Pi^2]);
```

$$t2 := \text{table}([\, 1 = 67832,\ 2 = cadena\ x,\ 5 = \pi^2\,])$$

De cualquier forma, sus elementos deben ser accesados a través de los indices asignados.

```
> t2[1];
```

$$67832$$

```
> t2[5];
```

$$\pi^2$$

```
> t2[2];
```

cadena x

Nótese que, en el caso de las tablas, el índice no representa la posición en la cual se encuentra el elemento, es decir, las tablas no son manejadas como listas. Dicho índice es usado únicamente como identificador para cada uno de los elementos. En caso de que al definir la tabla no se especifiquen índices, la función asigna números enteros, tomando en cuenta el orden en que fueron proporcionados los elementos. Por ejemplo:

```
> t3 := table([x, marzo, 98.431, cuatro + seis]);
```

$$t3 := \text{table}([\,1 = x,\ 2 = marzo,\ 3 = 98{,}431,\ 4 = cuatro + seis\,])$$

En este caso podemos acceder a los elementos usando como índice su posición dentro de la tabla:

```
> t3[1];
```

$$x$$

```
> t3[2];
```

$$marzo$$

```
> t3[3];
```

$$98{,}431$$

```
> t3[4];
```

$$cuatro + seis$$

Los elementos de una tabla pueden ser proporcionados también en forma de conjunto, pero en caso de que no se proporcionen índices (como en el ejemplo anterior), el índice de los elementos no necesariamente corresponde a la posición en la cual fueron colocados (tómese en cuenta que se está creando la tabla a partir de un conjunto y en este el orden es irrelevante). Por ejemplo:

```
> t3 := table({x, marzo, 98.431, cuatro + seis});
```

$$t3 := \text{table}([\,1 = x,\ 2 = marzo,\ 3 = cuatro + seis,\ 4 = 98{,}431\,])$$

Una vez creada una tabla podemos agregarle un elemento usando una instrucción como la siguiente:

```
> t3[5] := 24;
```

$$t3_5 := 24$$

Obviamente el elemento 5 no debe existir, en caso contrario modificariamos uno de los ya existentes.

En este tipo de estructuras, para poder visualizar todos los elementos es necesario utilizar la función **eval**:

```
> eval(t3);
```

$$\text{table}([\,1 = x,\ 2 = marzo,\ 3 = cuatro + seis,\ 4 = 98{,}431,\ 5 = 24\,])$$

Podemos modificar los elementos de una tabla de la misma forma que en el caso de las listas. Por ejemplo, modifiquemos el elemento indexado por **"2"**:

```
> t3[2] := junio;
```

$$t3_2 := junio$$

```
> eval(t3);
```

$$\text{table}([\,1 = x,\ 2 = junio,\ 3 = cuatro + seis,\ 4 = 98{,}431,\ 5 = 24\,])$$

Una forma de remover uno de los elementos es usando una ***"desasignación"***. Por ejemplo, eliminemos el elemento indexado por **"4"** en la tabla **t3**:

```
> t3[4] := 't3[4]';
```

$$t3_4 := t3_4$$

```
> eval(t3);
```

$$\text{table}([\,1 = x,\ 2 = junio,\ 3 = cuatro + seis,\ 5 = 24\,])$$

De esta manera podemos modificar una tabla previamente creada, eliminando o agregando elementos a los ya existentes.

2.5. Arreglos

Los arreglos son estructuras cuyos elementos se encuentran indexados de la misma forma que las tablas, pero en este caso los índices siempre son números enteros. Para este tipo de estructuras no es posible utilizar cadenas, u otro tipo de expresiones diferentes de números positivos, como índices. La forma en la cual se crean los arreglos es a través de la función **array**. Su sintaxis es:

array(dimensiones, elementos);

Donde dimensiones indica la cantidad de elementos que formarán el arreglo. Los elementos pueden o no proporcionarse en el momento de la creación del arreglo. Por ejemplo:

```
> arg1 := array(3..4, [4, Sistema]);
```

$$\begin{aligned} &arg1 := \text{array}(3.,4,\ [\\ &(3) = 4\\ &(4) = Sistema\\ &]) \end{aligned}$$

Esta instrucción crea un arreglo de dos datos, cuyos índices serán **“3"** y **“4"**, y sus elementos estarán dados por el número **“4"** y la cadena **“Sistema"**. Estos elementos pueden ser accesados utilizando sus índices:

```
> arg1[3];
```

$$4$$

```
> arg1[4];
```

$$Sistema$$

```
> arg1[2]; # obviamente este no es un elemento válido
```

```
Error, 1st index, 2, smaller than lower array bound 3
```

Nótese que, al igual que las tablas, los índices sirven como identificadores de los datos del arreglo, no necesariamente existe una relación entre el índice y la posición del elemento (como puede verse en la última instrucción). Veamos otro ejemplo:

```
> arg2 := array(2..5);
```

$$arg2 := \text{array}(2.,5,\ [])$$

En este caso creamos un arreglo pero sin asignarle elementos, esta asignación podemos hacerla posteriormente:

```
> arg2[2] := ‘cadena 1‘^2;
```

$$arg2_2 := cadena\ 1^2$$

Para visualizar los elementos de un arreglo es necesario utilizar la función **eval**.

```
> eval(arg2);
```

array(2.,5, [
(2) = *cadena* 1^2
(3) = $?_3$
(4) = $?_4$
(5) = $?_5$
])

Podemos ver que los últimos tres elementos aparecen indeterminados. Asignemos el resto de los datos:

```
> arg2[3] := sin(sqrt(2));
```

$$arg2_3 := \sin(\sqrt{2})$$

```
> arg2[4] := 785;
```

$$arg2_4 := 785$$

```
> arg2[5] := 87.32;
```

$$arg2_5 := 87{,}32$$

```
> eval(arg2);
```

array(2.,5, [
(2) = *cadena* 1^2
(3) = $\sin(\sqrt{2})$
(4) = 785
(5) = 87,32
])

En estas estructuras es posible desasignar un elemento pero éste no es eliminado del arreglo, solo su valor asignado. Por ejemplo:

```
> arg2[2] := 'arg2[2]';
```

$$arg2_2 := arg2_2$$

```
> eval(arg2);
```

array(2.,5, [
(2) = $?_2$
(3) = $\sin(\sqrt{2})$
(4) = 785
(5) = 87,32
])

Al igual que en las tablas, si no se proporcionan índices, la función **array** automáticamente asigna a cada elemento un número entero que corresponde a su posición, iniciando en uno. Por ejemplo:

```
> arg3 := array([exp(f(x)), 8*x, 3 - cos(w), 8754]);
```

$$arg3 := \left[e^{\mathrm{f}(x)},\ 8\,x,\ 3 - \cos(w),\ 8754 \right]$$

```
> arg3[1]; arg3[2]; arg3[3]; arg3[4];
```

$$e^{\mathrm{f}(x)}$$
$$8\,x$$

$$3 - \cos(w)$$

$$8754$$

Los arreglos también pueden ser creados con índices multiples, de la misma forma que una matriz de varias dimensiones (de hecho pueden usarse arreglos para crear matrices). Por ejemplo:

```
> arg4 := array(2..4, 2..4, [[1, 9, 8] , [7, 4, 93], [28, 5, 18]]);
```

$$\begin{aligned}
&arg4 := \text{array}(2.,4,\ 2.,4,\ [\\
&(2,\ 2) = 1\\
&(2,\ 3) = 9\\
&(2,\ 4) = 8\\
&(3,\ 2) = 7\\
&(3,\ 3) = 4\\
&(3,\ 4) = 93\\
&(4,\ 2) = 28\\
&(4,\ 3) = 5\\
&(4,\ 4) = 18\\
&])
\end{aligned}$$

En este caso, los elementos deben ser accesados por medio de dos índices.

```
> arg4[2, 2];
```

$$1$$

```
> arg4[3, 2];
```

$$7$$

```
> arg4[4, 4];
```

$$18$$

Este tipo de datos también pueden ser usados para crear matrices o vectores, de tal manera que puedan ser operados con matrices o vectores creados con la función **Matrix** o **Vector**, respectivamente, o bien por funciones del paquete **LinearAlgebra**. Para poder usarlos de esta forma debemos convertirlos previamente por medio de la instrucción **convert**. Por ejemplo:

```
> Ma := Matrix(3, 3, [[5, 4, 9], [6, 18, 2], [8, 23, 16]]);
```

$$Mat := \begin{bmatrix} 5 & 4 & 9 \\ 6 & 18 & 2 \\ 8 & 23 & 16 \end{bmatrix}$$

```
> Ar := array([[1, 2, 3], [Pi, 2, sin(Pi/2)], [-3, 2, 8]]);
```

$$Ar := \begin{bmatrix} 1 & 2 & 3 \\ \pi & 2 & 1 \\ -3 & 2 & 8 \end{bmatrix}$$

```
> Mb := convert(Ar, Matrix);
```

$$Mb := \begin{bmatrix} 1 & 2 & 3 \\ \pi & 2 & 1 \\ -3 & 2 & 8 \end{bmatrix}$$

```
> Ma.Mb;
```

$$Ma \; . \; \begin{bmatrix} 1 & 2 & 3 \\ \pi & 2 & 1 \\ -3 & 2 & 8 \end{bmatrix}$$

```
> Ma + Mb;
```

$$Ma + \begin{bmatrix} 1 & 2 & 3 \\ \pi & 2 & 1 \\ -3 & 2 & 8 \end{bmatrix}$$