MateFun - Manual

Sylvia da Rosa, darosa@fing.edu.uy[1], Marcos Viera, mviera@fing.edu.uy[1], and Juan Pablo García, jpgarcia@fing.edu.uy[1]

[1]Instituto de Computación - Facultad de Ingeniería - Universidad de la República

I. Introducción

El lenguaje MateFun surgió como un proyecto del Centro Interdisciplinario para la Cognición la Enseñanza y el Aprendizaje (CICEA) del Espacio Interdisciplinario de la UDELAR. Es un lenguaje de programación funcional pura dirigido al aprendizaje de funciones matemáticas, diseñado por investigadores del Instituto de Computación (InCo) de la Facultad de Ingeniería (FING) UDELAR. MateFun puede ser accedido a través de un entorno de programación integrado web[1] que permite gestionar programas, programar, ejecutar programas y visualizar gráficas, figuras y animaciones. La sintaxis del lenguaje fue diseñada con el objetivo de ser minimal y lo más parecida posible a la notación utilizada en matemáticas. La idea es que el lenguaje se pueda asimilar rápidamente y que su relación con los conceptos matemáticos subyacentes pueda ser reconocida por los estudiantes. Considerando que el público objetivo está compuesto por estudiantes de secundaria de habla hispana, las palabras clave se definieron en español[2].

Un programa en MateFun es una lista de definiciones de **conjuntos** y **funciones**.

En las siguientes secciones se presentan los elementos básicos de MateFun.

I-A. Conjuntos

Los conjuntos se utilizan para determinar el dominio y codominio de las funciones (corresponden a los tipos de los lenguajes de programación).

```
conj Mes = { Enero , Febrero , Marzo
           , Abril , Mayo , Junio
           , Julio , Agosto , Setiembre
           , Octubre , Noviembre , Diciembre }

conj Rno0 = { x en R | x /= 0 }

conj Rango = {x en Z | (x > 2, x < 10)}
```

Figura 1. Ejemplos de conjuntos definidos por el usuario

En la Figura 1 se muestran algunos ejemplos de definiciones de conjuntos. Un conjunto se define usando la palabra reservada **conj**, se le asigna un nombre y los elementos que contiene.

El nombre de un conjunto es una cadena de caracteres alfanuméricos que comienza con un caracter alfabético en mayúscula.

Los elementos se pueden expresar por enumeración o por comprensión. Para definir un conjunto por enumeración se deben enumerar, separados por coma, los elementos (cadenas alfanuméricas que comienzan con mayúscula) que lo integran.

Por ejemplo, Mes en la Figura 1 es un conjunto definido por enumeración que contiene los elementos Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Setiembre, Octubre, Noviembre y Diciembre.

Para definir un conjunto por comprensión se debe establecer un conjunto de partida y la condición que deben cumplir los elementos de dicho conjunto para pertenecer al conjunto que se está definiendo. Por lo tanto el nuevo conjunto será subconjunto del conjunto de partida.

Una condición puede ser una relación binaria entre dos expresiones o una lista de estas relaciones, entre paréntesis y separadas por comas, como se muestra en los conjuntos Rno0 y Rango de la Figura 1.

Las relaciones binarias son las usuales de igualdad y orden. Notar que la relación de igualdad se indica con ==.

```
"==" , "/=" , "<" , ">" , "<=" , ">="
```

En el caso de tener una lista de relaciones, el resultado de su evaluación es la conjunción lógica ($\wedge$) entre todas ellas. El conjunto Rno0 de la Figura 1 representa al conjunto de los reales (**R**) distintos de 0, mientras que el conjunto Rango representa al subconjunto de (N) compuesto por $\{3, 4, 5, 6, 7, 8, 9\}$. Notar que las variables, a diferencia de los nombres y elementos de conjuntos, comienzan con minúsculas.

Los conjuntos pueden ser elementales o compuestos. Los conjuntos elementales son los primitivos (**R** para los reales, **Z** para los enteros, **Fig** para las figuras en dos dimensiones, **Fig3D** para las figuras en tres dimensiones y **Color** para los colores) y los enumerados definidos por el usuario. También son elementales los conjuntos definidos por comprensión que tienen como conjunto de partida un conjunto elemental. Los conjuntos compuestos son las n-tuplas de conjuntos y las secuencias de un conjunto. La n-tupla es la generalización del producto cartesiano. Por ejemplo la tupla **R** X **R** contiene pares de reales, que se escriben entre paréntesis y separados por coma; un elemento de este conjunto es (2.3,4.2) . La secuencia es lo que en los lenguajes de programación se suele llamar lista; es decir, el conjunto de las secuencias de

[1]https://www.fing.edu.uy/proyectos/matefun/

[2]Es posible configurarlo para otros idiomas. Actualmente se cuenta con una versión en inglés.

```
cuad :: R -> R
cuad(x) = x * x

inverso :: Rno0 -> R
inverso(x) = 1 / x

areaTria :: R X R -> R
areaTria(base,alt) = (base * alt) / 2

pi :: () -> R
pi () = 3.1416

areaCirc :: R -> R
areaCirc(r) = pi() * cuad(r)
```

Figura 2. Ejemplos de funciones definidas por el usuario

elementos de un conjunto dado. Una posible secuencia de reales (**R***) es 1.8:9.3:2.4:[].

I-B. Funciones

Para definir una función se debe indicar su **signatura** y la **ecuación** que la define.

La signatura se compone del nombre de la función, el conjunto dominio y el codominio.

Por ejemplo, la signatura de la función cuad de la Figura 2 es cuad :: **R** ->**R**, indicando que la función cuad va de reales a reales. Notar que en inverso utilizamos como dominio el conjunto Rno0 definido en la Figura 1. En la signatura de la función areaTria podemos ver que para representar funciones de múltiples variables (en este caso dos reales) utilizamos tuplas como dominio. También es posible definir la operación nularia, que va del producto vacío a A, o sea () -> A lo que permite definir y usar constantes, como Pi en la Figura 2.

La ecuación se define dando el nombre de la función, las variables independientes (es decir, parámetros) y el cuerpo de la función. El cuerpo de una función se compone de una expresión o de una lista de expresiones.

En el caso de la función cuad, definida en la Figura 2, la variable independiente es x y el cuerpo de la función es la expresión x * x.

En el caso de que el cuerpo sea una lista de expresiones, cada expresión se ejecuta al cumplirse una condición (si no se cumple ninguna de las anteriores). La lista finaliza con una expresión por defecto, que se ejecuta si no se cumple ninguna de las condiciones. Por ejemplo, en la función abs, definida en la Figura 3, la variable independiente es x y el cuerpo de la función es la lista de las expresiones x (se ejecuta si x >= 0) y -x (se ejecuta si no se cumple x >= 0).

Las condiciones de cada caso se analizan en el orden en que están definidos (de "arriba" hacia "abajo") y una vez que se cumple una de ellas, se evalúa la expresión asociada y no se analizan más casos. Para asegurar que las funciones sean totales en el dominio, el lenguaje impone la existencia de un caso por defecto.

```
abs :: R -> R
abs(x) =  x si x >= 0
       { -x

max :: R X R -> R
max(x,y) = x si x >= y
         { y

dias :: Mes -> R
dias(m) = 31 si m == Enero
        { 28 si m == Febrero
        { 31 si m == Marzo
        { 30 si m == Abril
        { 31 si m == Mayo
        { 30 si m == Junio
        { 31 si m == Julio
        { 31 si m == Agosto
        { 30 si m == Setiembre
        { 31 si m == Octubre
        { 30 si m == Noviembre
        { 31
```

Figura 3. Ejemplos de funciones definidas por casos

La función dias retorna la cantidad de días que (generalmente) tiene un mes. Hay dos aspectos importantes a destacar sobre la función dias que tienen que ver con las decisiones de diseño del lenguaje. El primero es que la función no es numérica, esto es importante porque uno de los objetivos de enseñar funciones mediante la programación es reforzar la idea de que no todas las funciones son numéricas. El otro es la *verbosidad* de la función, que es un precio a pagar debido a la simplicidad del lenguaje; en este caso por la falta de expresiones condicionales (del tipo if-then-else) y operadores booleanos. Esto no nos resulta particularmente preocupante, dado que priorizamos el rápido aprendizaje en lugar de la utilidad del lenguaje para la implementación de aplicaciones de mediano o gran porte[3].

Un valor literal puede ser un elemento de un conjunto definido por enumeración, un número, un color, una figura vacía o una secuencia vacía.

Los operadores son los operadores aritméticos (definidos para **Z**, **R** y todos sus sub-conjuntos) ("+","−","*","/",""), más la proyección de una tupla (**!**) y el insertar al inicio de una secuencia (**:**).

Una expresión puede ser también la aplicación de una función en cuyo caso consiste del nombre de la función y entre paréntesis separados con comas sus argumentos, por ejemplo (max(3,7)). Las funciones pueden ser las declaradas por el usuario o las funciones primitivas, definidas para operar con números, secuencias, colores y figuras. Por ejemplo, en el cuerpo de la función areaCirc de la Figura 4 se aplica

[3]Ejercicio para el lector: definir el conjunto Bool con los valores V y F y las funciones correspondientes para que la función dias conste de un número reducido de ecuaciones.

```
factorial :: N -> N
factorial(x) = x * factorial(x-1) si x > 0
             { 1

f :: R -> R
f (x) = x^3
g :: R -> R
g (x) =  x - 1
h :: R -> R
h (x) = g . f (x)
```

Figura 4. Ejemplos de definiciones de funciones usando otras funciones y composicion

la función cuad a la variable r. Por lo tanto, esta función calcula el área de un triángulo dado su radio r. La función factorial tiene una definición recursiva, es decir que se invoca a sí misma.

Además de los operadores aritméticos, el lenguaje provee de funciones primitivas para hallar el seno (sen), coseno (cos) y raíz cuadrada (raizcuad) de un número.

I-B1. Definiciones locales: Una definición local define una variable dentro de la definición de una función. En el primer ejemplo abajo, se define una función **raices** para hallar las raices reales de una ecuación de segundo grado, definiendo delta localmente, como suele hacerse en matemática (para simplificar el primer ejemplo, se supone que la ecuación tiene raíces reales). En el segundo ejemplo se define **divEntera** que toma dos naturales, el segundo distinto de cero, y devuelve cociente y resto de la división entera del primero por el segundo.

```
raices :: R X R X R -> R X R
raices (a,b,c) = ((-b+delta)/(2*a),
                  (-b-delta)/(2*a))
    donde delta = raizcuad(b^2- 4*a*c)

divEntera :: N X Nno0 -> N X N
divEntera (a,b) = (c,r)
    donde c = div(a,b)
          r = mod (a,b)
```

Figura 5. Ejemplos de definiciones locales

I-C. Secuencias

Para poder manipular *colecciones de elementos* de una forma simple, el lenguaje incorpora la noción de secuencia. Las secuencias se definen de manera inductiva, con el valor **[]** de secuencia vacía y el operador **:** que a partir de un elemento y una secuencia de elementos (todos pertenecientes al mismo conjunto), retorna la secuencia resultante de agregar el elemento al principio de la secuencia dada.

```
suma :: R* -> R
suma(l) = 0 si l == []
        { primero(l) + suma(resto(l))

largo :: R* -> R
largo(l) = 0 si l == []
         { 1 + largo(resto(l))


conj RSeqNV = { l en R* | largo(l) /= 0 }

maximo :: RSeqNV -> R
maximo (l) = primero(l) si resto(l) == []
           { max( primero(l)
                , maximo(resto(l)))
```

Figura 6. Ejemplos de funciones recursivas sobre secuencias

Por ejemplo, utilizando secuencias podemos definir la función "tienen", que dado un número de días, retorna todos los meses que tienen en total ese número de días (y la secuencia vacía si ningún mes los tiene):

```
tienen :: N -> Mes*
tienen(d) = [Abril, Junio, Setiembre,
             Noviembre] si    d == 30
          { [Enero, Marzo, Mayo, Julio,
             Agosto, Octubre,
             Diciembre] si    d == 31
          { [Febrero]    si ( d == 28
                            , d == 29)
          { []
```

Además de esa forma de construcción de secuencias, la función **rango** permite construir secuencias de racionales a partir de un valor inicial, un valor final y un valor de paso. Por ejemplo **rango**(0,100,10) retorna la secuencia [0,10,20,30,40,50,60,70,80,90,100] .

Para poder obtener los elementos de una secuencia existen las funciones *destructoras* **primero** y **resto**, que respectivamente retornan el primer elemento de una secuencia y la secuencia resultante de quitar el primer elemento. Por ejemplo la expresión **primero**(**rango**(0,100,10)) retorna el valor real 0 y **resto**(**rango**(0,100,10)) retorna la secuencia 10**:**20**:**30**:**40**:**50**:**60**:**70**:**80**:**90**:**100**:[]**. Notar que en estos ejemplos hemos realizado *composición de funciones*, dado que a **primero** y **resto** les hemos pasado el resultado de aplicar la función **rango**. Es decir, hemos usado la expresión derecha de la definición de composición de funciones, que es: f . g (x) = f (g (x)), donde los dominios y codominios de f y g cumplen determinadas condiciones. También podemos usar la expresión izquierda y escribir por ejemplo, **primero** . **rango** (0,100,10) .

Dada la naturaleza inductiva del conjunto secuencia, es natural que muchas funciones que operan sobre secuencias se definan usando recursión. En la Figura 6 se muestran

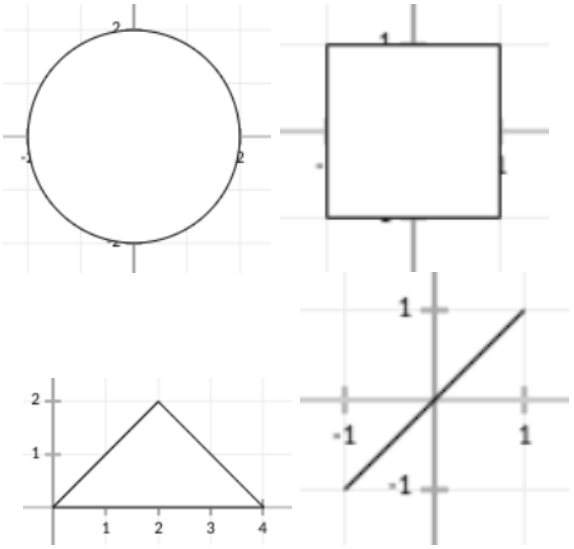

Figura 7. Ejemplos de figuras 2D

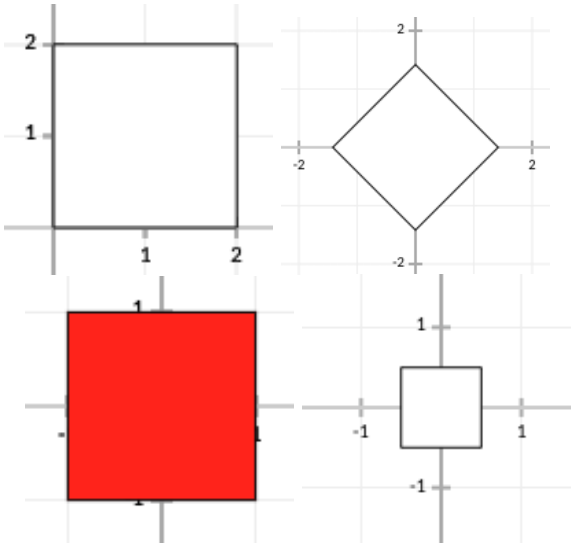

Figura 8. Transformaciones de figuras

algunos ejemplos de este tipo de funciones. La función suma, para obtener la suma de una secuencia de reales, suma el primer real de la secuencia con el resultado de la suma del resto (considerando que la secuencia vacía suma 0). De forma similar se puede calcular el largo de una secuencia. Para la función maximo, que calcula el máximo de una secuencia de reales no vacía, definimos un nuevo conjunto RSeqNV, de manera que la misma no sea parcial en su dominio. Notar que dentro de las condiciones de los conjuntos se pueden utilizar funciones definidas en el mismo programa.

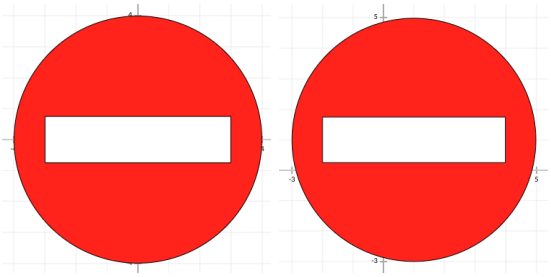

Figura 9. Juntar figuras

I-D. Colores, Figuras y Animaciones

En MateFun es posible definir funciones que permiten la creación y manipulación de figuras. Además de los colores predefinidos, se pueden obtener nuevos colores utilizando la función **rgb**, que a partir de tres números entre 0 y 1 que indican respectivamente el aporte de rojo, verde y azul (*red*, *green*, *blue*), compone el color resultante.

Las figuras pueden crearse en dos dimensiones (figuras 2D) o en tres dimensiones (figuras 3D).

La función **aFig** retorna una figura con el texto de un enumerado, **rect** retorna un rectángulo a partir de su base y altura, **circ** retorna un rectángulo a partir de su radio, **segmento** retorna un segmento de recta dados dos puntos (**R** X **R**) y **poli** retorna un polígono que une la secuencia de puntos (**R**∗) que recibe como parámetro. Salvo **segmento** y **poli**, las demás figuras se crean siempre en el centro (0,0) de un sistema de coordenadas cartesianas. En la Figura 7 se muestran las figuras generadas por las expresiones **circ** (2), **rect** (2,2), **poli** ((0,0) **:** (2,2) **:** (4,0) **: []**) y **segmento**((−1,− 1),(1,1)) . Dos figuras se pueden unir en una nueva figura utilizando la función **juntar**. A una figura se la puede pintar con un color dado utilizando la función **color**. También a una figura se le puede mover, rotar y escalar utilizando las funciones homónimas. En la Figura 8 se muestran las figuras generadas por las expresiones **mover**(**rect** (2,2),(1,1)) , **rotar**(**rect** (2,2),45) , **color**(**rect** (2,2), **Rojo**) y **escalar**(**rect** (2,2),0.5) .

En la Figura 9 se muestra las figura generada por la expresión **juntar**(**color**(**circ** (4), **Rojo**), **rect** (6,1.5)) , que une un círculo rojo con un rectángulo. También se muestra el efecto de mover esa figura al punto (1,1) haciendo **mover**(**juntar**(**color**(**circ** (4), **Rojo**), **rect** (6,1.5)),(1,1)) .

En MateFun, una animación se define como una secuencia de figuras. Entonces por ejemplo la siguiente función produce una animación que rota una figura dada de a 10 grados tantas veces como se le indique.

```
rotFig :: Fig X R -> Fig*
rotFig(fig, cant)
    = [] si cant <= 0
    { fig : rotFig(rotar(fig,10),cant-1)
```

Usando la tupla vacía como dominio se pueden definir expresiones, figuras o animaciones constantes. Por ejemplo, al definir relojArena como se muestra en la Figura 11 se obtiene la figura en 3D mostrada en la Figura 10 con la expresión relojArena().

I-E. El Intérprete

MateFun es un lenguaje *interpretado*, lo que significa los programas "se cargan" para ser ejecutados por un intérprete. Al cargar un programa todas sus definiciones quedan disponibles en el intérprete para ser utilizadas en su línea de comandos.

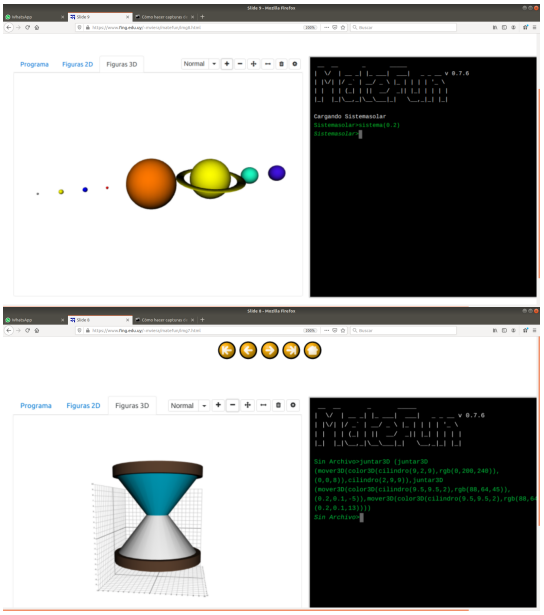

Figura 10. Ejemplos de figuras 3D

```
figur :: R X R X R X R X R X R-> Fig3D
figur(x,y,z,h,p,t)=color3D(cilindro(x,y,z),
                           rgb(h,p,t))

relojArena :: () -> Fig3D
relojArena()=juntar3D (juntar3D (mover3D
        (figur(9,2,9,0,240,200),(0,0,8)),
        cilindro(2,9,9)),
        juntar3D (mover3D(figur
        (9.5,9.5,2,88,64,45),(0.2,0.1,-5)),
        mover3D(figura1(9.5,9.5,2,88,64,45),
        (0.2,0.1,13))))
```

Figura 11. Ejemplos de uso de dominio vacio

Por ejemplo, consideremos que los códigos de las figuras 1, 2, 3, 4 y 6, forman un programa llamado Ejemplos. Al cargarlo en el intérprete tendríamos disponibles, además de las funciones, operadores y conjuntos primitivos, todas las funciones y conjuntos que se definieron en esas figuras.

El comando más básico del intérprete consiste en evaluar expresiones, las cuales pueden hacer uso de todas las definiciones disponibles. La Figura 12 es un ejemplo de sesión válida. El entorno de programación integrado web provee de una ventana gráfica en la cual se exhiben las figuras y animaciones en las que resulten las expresiones de este tipo. Las Figuras 7, 8 y 9 son capturas de esta ventana.

Para el caso de funciones numéricas y no numéricas de una sola dimensión (es decir, con dominio y codominio **R** o un subconjunto de **R**) se puede utilizar esa ventana para visualizar sus gráficas. Esto se realiza con el comando **?grafica** del intérprete. Entonces al ejecutar

```
Ejemplos>?grafica cuad
```

Figura 12. Intérprete

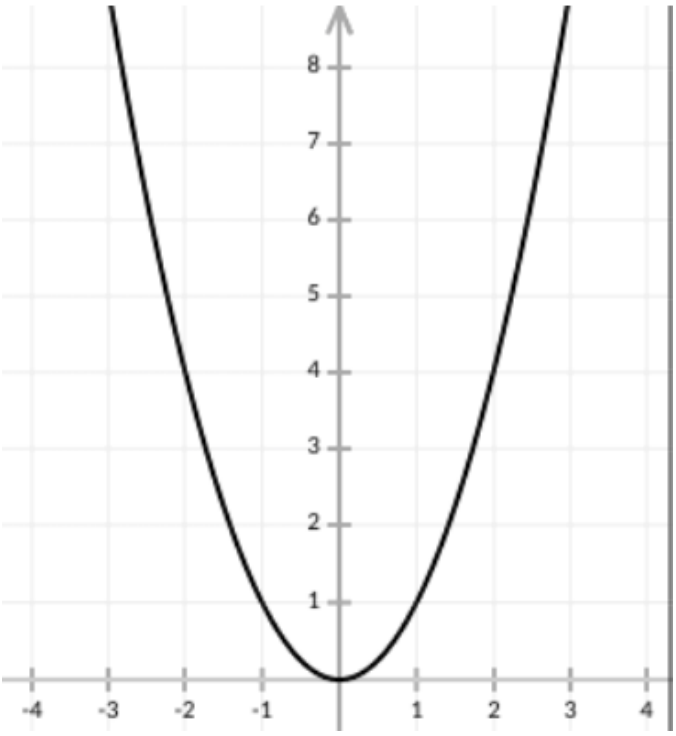

Figura 13. Gráfica de función

se obtiene la gráfica de la Figura 13.

I-F. Comandos del intérprete

Los siguientes comandos permiten realizar las acciones que describe cada uno.

```
>?ayuda
Comandos del Interprete
!salir                  salir
!cargar <programa>      cargar un programa
!recargar
volver a cargar el programa actual
?funs                   lista las funciones
?vars
lista las variables y sus valores
```

```
?conjs                  lista los conjuntos
?fun <funcion>
despliega el dominio y codominio de una funcion
?var <variable>
despliega el valor de una variable
?conj <conjunto>
despliega la definicion del conjunto
?grafica <funcion>
grafica una funcion de R -> R
?ayuda
despliega este mensaje de ayuda
```

Por ejemplo, el comando **?funs** despliega una lista con la signatura de cada función disponible en el intérprete, como se muestra a continuación:

```
>?funs
- :: R -> R
red :: R -> Z
sen :: R -> R
cos :: R -> R
raizcuad :: R -> R
rgb :: (R X R X R) -> Color
rect :: (R X R) -> Fig
circ :: R -> Fig
segmento :: ((R X R) X (R X R)) -> Fig
poli :: (R X R)* -> Fig
juntar :: (Fig X Fig) -> Fig
color :: (Fig X Color) -> Fig
mover :: (Fig X (R X R)) -> Fig
rotar :: (Fig X R) -> Fig
escalar :: (Fig X R) -> Fig
aFig :: A -> Fig
segmento3D :: ((RXRXR) X (RXRXR))->Fig3D
esfera :: R -> Fig3D
cilindro :: (R X R X R) -> Fig3D
cubo :: (R X R X R) -> Fig3D
anillo :: (R X R X R) -> Fig3D
juntar3D :: (Fig3D X Fig3D) -> Fig3D
color3D :: (Fig3D X Color) -> Fig3D
mover3D :: (Fig3D X (R X R X R)) -> Fig3D
rotar3D :: (Fig3D X (R X R X R)) -> Fig3D
escalar3D :: (Fig3D X R) -> Fig3D
rango :: (R X R X R) -> R*
primero :: A* -> A
resto :: A* -> A*
```

Observar que hay funciones para las figuras 2D (por ejemplo, juntar, mover) y para las figuras 3D (por ejemplo juntar3D, mover3D). Componiendo dichas funciones es posible crear figuras como las de las Figuras 7, 8, 9 y 10 de este manual.

I-G. Errores y advertencias

Al cargar un programa en el intérprete se realiza un chequeo de errores. En el caso de que haya alguno, se despliega el mensaje correspondiente y la línea/columna del texto en el que ocurrió, y el programa no se ejecuta. Todos los textos de los mensajes son en español y refieren a errores conceptuales.

Supongamos que tenemos la función areaTria de la Figura 2 definida en un archivo llamado 'Ejemplos', y definimos una función 'areaRara' así

```
areaRara :: R -> R
areaRara(x) = areaTria(x)
```

Al cargar 'Ejemplos' en el intérprete se produce el error

```
Error: {archivo: Ejemplos linea: 53 columna: 25}
 Se esperan elementos de (R X R)
 pero se encontro R.
```

También hay advertencias, que no impiden la ejecución del programa pero cuyo origen debe ser revisado porque puede ser fuente de errores futuros. O sea que las advertencias informan posibles problemas que en esa etapa no se han podido todavía detectar como errores. Si en el archivo 'Ejemplos' definimos la función

```
mismo :: R -> Rno0
mismo(x) = x
```

se despliega una advertencia al cargarlo en el intérprete

```
Advertencia: {archivo: Ejemplos linea: 3 columna:
Conjunto Rno0 requerido es subconjunto del resulta
 Por lo que existe la posibilidad de que su valor
fuera del conjunto.
```

y un error al aplicarla a 0, como se muestra a continuación:

```
mismo(0)

Error: {archivo: Ejemplos linea: 3 columna: 12}
 Valor 0 no pertenece al conjunto Rno0
porque no se cumple: [0 /= 0].
```