TUTORIAL DE PROGRAMACIÓN HP USER RPL EN MODO ALGEBRAICO

Versión 1.2

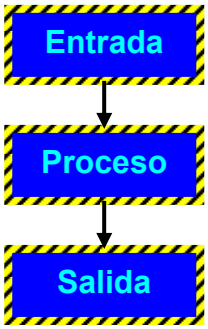
Por David Enrique Torres

ÍNDICE

INTRODUCCIÓN

Programar en modo algebraico no se parece a otra forma de programación. La diferencia entre el modo RPN y el algebraico es que en modo algebraico las funciones o comandos se escriben primero y después, los argumentos; y en el modo RPN, al revés. Ésta es una ventaja del modo algebraico si se tiene que escribir una operación tremendamente complicada, pues se escribiría como en papel.
Es indispensable saber manejar las listas, dado que si un comando o función proporciona varias respuestas, éstas aparecen agrupadas en una lista, la cual a veces contiene a otra lista.

La manera en que fluye un programa es la siguiente:



Lo último que se escribe es la salida.
Los comentarios se escriben con una arroba (@), pero al compilar el programa los comentarios desaparecen.

Ejemplo: Programa que eleva un número a la quinta potencia.

```
« → X
« X^5; 5+" DOLARES"
»
» ► QUINTA
```

Para ejecutar el programa se escribe su nombre y entre paréntesis se escriben los argumentos de entrada. Si se quiere elevar 2 a la quinta potencia:

QUINTA(2) **[ENTER]**

Al ejecutar el programa **QUINTA** se eleva el número introducido a la quinta potencia, pero como lo que se escribió por último fue el **5** más la cadena **" DOLARES"**, el resultado siempre será la cadena **"5 DOLARES"**.
El punto y coma (o sólo coma) se usa para separar las instrucciones del programa.
Lo último que se escribe no necesita tener punto y coma después.
Para tener una respuesta numérica no hay que escribir nada más después de X^5 en el código fuente del programa.

Un programa podría no tener entradas. También podría no tener una salida en forma numérica o una expresión matemática o un texto. En tal caso aparece la palabra NOVAL.
Ejemplo: Programa que hace sonar la bocina a 1000 hertz durante 2 segundos.

```
« BEEP(1000,2) » ► SONIDO
```

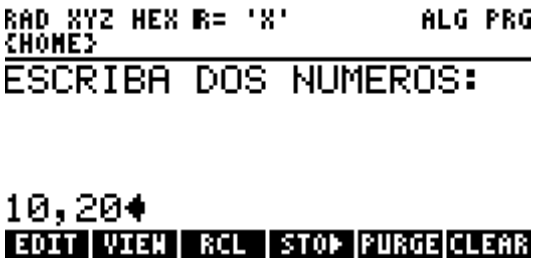
ENTRADA

Los anteriores han sido programas de tipo función, pero también es posible hacer programas de tipo guión. Para ello hay dos comandos: INPUT e INFORM.
CHOOSE es un comando que sirve para crear cuadros de selección.
No hay necesidad de escribir los datos de entrada entre paréntesis cuando se ejecuta un programa de tipo guión. Sólo se escribe el nombre del programa y se presiona la tecla [ENTER]

INPUT

La sintaxis es la siguiente: INPUT(“Texto”,“Texto opcional”)
La salida de INPUT es una cadena conteniendo lo que se ha introducido.

Ejemplo:
INPUT(“ESCRIBA DOS NÚMEROS:”,“”) [ENTER]
Los datos introducidos se separan por comas o espacios. Suponiendo que se introducen el 10 y el 20 se haría así:
10,20 [ENTER]



La salida sería una cadena conteniendo al 10 y al 20.
“10,20”

Como esto es inútil se hace uso del comando OBJ→.
Ejemplo:
OBJ→(INPUT(“ESCRIBA DOS NÚMEROS”,“”)) [ENTER]
Si se introducen el 10 y el 20 la salida será una lista conteniendo al 10 y al 20:
{10,20}

También es posible agregar texto opcional para mejorar la estética y evitar errores.
Ejemplo:

```
« ∏LIST(OBJ→(INPUT(“VOLUMEN DE UN CUBO”, {“:ALTO: ↵
  :ANCHO: ↵
  :LARGO:”,{1,0}})))
» ▶CUBO
```

Las variables se deben escribir como objetos etiquetados (con dos puntos a la izq. y dos puntos a la der.). La lista {1,0} indica que se posicionará el cursor inicialmente en la primera variable, y el 0 es un margen.
Después de escribir cada dato se presiona [▼] para bajar. Después de escribir el último número se presiona [ENTER].
El comando ∏LIST se usa para multiplicar todos los elementos contenidos en una lista.
↵ ([R-Shift] [.]) sirve para pasar al renglón siguiente y es como presionar [ENTER] al escribir un archivo de texto en WORD o en bloc de notas.

Ejemplo: Ídem, pero mostrando las variables en forma horizontal.

```
« ∏LIST(OBJ→(INPUT("VOLUMEN DE UN CUBO",
  {":ALTO:    :ANCHO:    :LARGO:",{1,0}}))
» ▶CUBO
```

INFORM

La sintaxis de INFORM es la siguiente:
INFORM("Título",{{"Variable 1:","Ayuda opcional",tipo de variable},
{"Variable 2:", "Ayuda opcional", tipo de variable},
{"Variable N:", "Ayuda opcional", tipo de variable}},
{Número de columnas, margen}, {Valores de reset},{valores iniciales})

Una variable podría ser de varios tipos. En dado caso se separarían por comas los tipos.
Un número típico para el margen podría ser 0, 1 ó 2.
Los valores de reset se escriben (opcionalmente) uno por cada variable.
Si se presiona el botón [NXT] cuando aparece el cuadro de diálogo, aparece el menú [RESET] [CALC] [TYPE] [ ] [CANCL] [OK]
Los valores iniciales se escriben (opcionalmente) uno por cada variable. Dichos valores aparecen al inicio en el cuadro de diálogo.
Si se presiona [TYPE] aparece la lista de tipos al que puede pertenecer el objeto que se digitará.

La salida de INFORM es una lista que contiene la lista de lo que se introdujo y el número 1 (indicando que se escribió algo).

Ejemplo: Cuadro de diálogo que tiene una columna, tres variables, valores iniciales (0, 0, 0) y valores de reset (1, 2, 3).

```
INFORM("TÍTULO",{{"X:","DIGITE UN NÚMERO",0,9},
{"Y:","DIGITE UN NÚMERO",0,9},
{"Z:","DIGITE UN NÚMERO",0,9}},
{1,0},{1,2,3},{0,0,0})
```

Ejemplo: Ídem, pero con tres columnas, margen más grande (2), sin valores iniciales ni de reset.

```
INFORM("TITULO",{{"X:","DIGITE UN NUMERO",0,9},
{"Y:","DIGITE UN NÚMERO",0,9},
{"Z:","DIGITE UN NÚMERO",0,9}},
{3,2},{ },{ })
```

La ayuda se escribe si es realmente necesaria. Entre más pequeño sea un programa, más rápido correrá (aunque es una diferencia de pocos segundos).

Suponiendo que se digitan 10 para X, 20 para Y, y 30 para Z, la salida del último ejemplo es:

```
{{10,20,30},1}
```

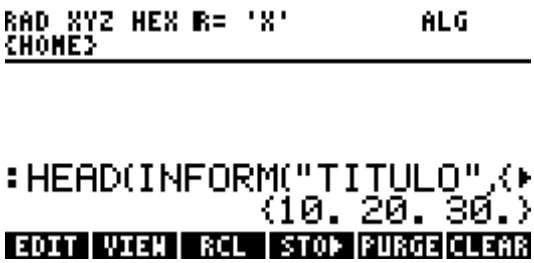
Como sólo interesa la lista interior se hace uso del comando HEAD, el cual obtiene el primer valor de una lista.

Ejemplo: El mismo ejemplo anterior usando HEAD.

```
HEAD(INFORM("TÍTULO",{{"X:","DIGITE UN NÚMERO",0,9},
{"Y:","DIGITE UN NÚMERO",0,9},
{"Z:","DIGITE UN NÚMERO",0,9}},
{1,0},{1,2,3},{0,0,0}))
```

La salida será:

```
{10,20,30}
```



El tipo de variable aparece en la página 18 de la Guía de HP 49G Pocket. A continuación se presentan algunos tipos de variables.

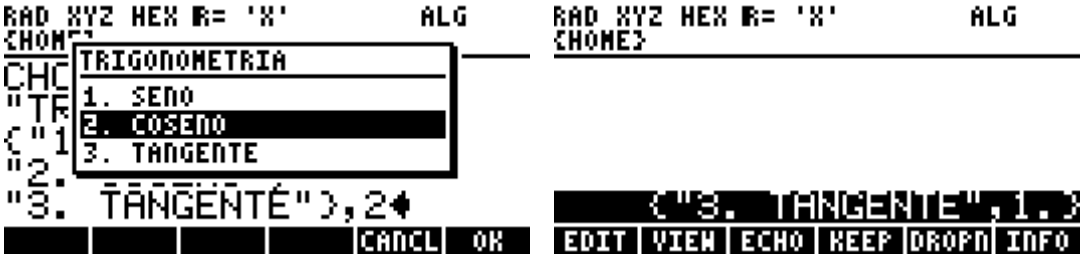
NÚMERO	TIPO	EJEMPLO
0	Número real	-1.025
1	Número complejo	5+2*i
2	Cadena	"HP 49G"
3	Sistema de números reales	[[1,2],[3,4]]
4	Sistema de números complejos	[[1,2*i],[3+i,-5-10i]]
5	Lista	{1,2,3,4,5}
6	Nombre global	X
9	Objeto algebraico	'pi'
18	Función incorporada	SIN
19	Comando incorporado	INPUT
28	Entero	3
29	Vector, matriz simbólica	[X,8,Z]

CHOOSE

La sintaxis es la siguiente: CHOOSE("Título",{Variable 1, Variable 2, Variable N}, Número de posición inicial)
La salida de CHOOSE es una lista conteniendo lo que se seleccionó y el número 1 (indicando que se seleccionó algo).

Ejemplo: Cuadro de selección de funciones trigonométricas. Al inicio está seleccionada la segunda opción.

```
CHOOSE("TRIGONOMETRÍA",
{"1. SENO","2. COSENO","3. TANGENTE"},2)
```



Suponiendo que se elige la tercera opción, la salida será:

```
{"3. TANGENTE", 1}
```

Para obtener sólo lo que se ha seleccionado se usa el comando GET. La sintaxis de GET es: GET(Lista o vector o matriz, Número de elemento seleccionado). Luego se usa el comando OBJ→.

```
OBJ→(GET(CHOOSE("TRIGONOMETRÍA",{"1. SENO", "2.
COSENO", "3. TANGENTE"},2),1))
```

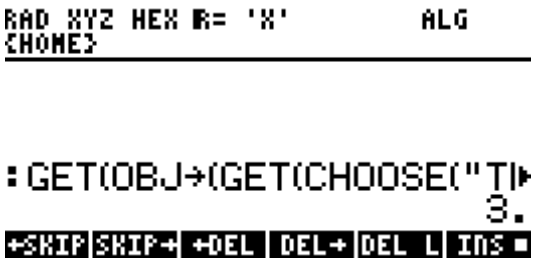
Primero se selecciona el primer elemento de la lista de salida de CHOOSE, el cual es de tipo cadena ("3. Tangente"). Luego se aplica el comando OBJ→ y se obtiene una lista que contiene el número seleccionado y el nombre en forma simbólica (como si fuera una variable). Como lo que realmente importa es el número seleccionado, se vuelve a aplicar el comando GET.

```
GET(OBJ→(GET(CHOOSE("TRIGONOMETRÍA",{"1. SENO", "2.
COSENO", "3. TANGENTE"},2),1)),1)
```

También es posible utilizar el comando HEAD en lugar de GET.

```
HEAD(OBJ→(HEAD(CHOOSE("TRIGONOMETRÍA",{"1. SENO", "2.
COSENO", "3. TANGENTE"},2))))
```

La salida es ahora el número 3. Luego se aplicaría una bifurcación o una iteración para realizar alguna acción.



WAIT

Sirve para hacer una pausa. La sintaxis es: WAIT(Número de segundos)
No tiene salida.
Si el argumento es 0 ó un número negativo, la pausa se extiende un tiempo indefinido hasta que se presione cualquier tecla (excepto R-Shift, L-Shift, ALPHA y ON), y la salida de WAIT es la posición de la tecla presionada.

SALIDA

Para obtener en la salida dos o más valores se usan listas o vectores, principalmente las listas porque permiten el uso de cadenas.
Ejemplo: Programa que eleva un número al cubo, a la cuarta y quinta potencia.

```
« → X
« {X+"^3= "+X^3," X+"^4= "+X^4," X+"^5= "+X^5}
»
» ▶POTENCIA
```

Suponiendo que se introduce el 5, la salida será:

```
{"5^3= 125","5^4= 625","5^5= 3125"}
```

Ejemplo: Ídem, usando INPUT.

```
« OBJ→(INPUT("Digite un número:","")) → X
« {X+"^3= "+X^3, X+"^4= "+X^4, X+"^5= "+X^5}
```

```
»
» ►POTENCIA
```

Ejemplo: Programa que calcula el volumen y la superficie de un paralelepípedo. Para obtener el n-ésimo elemento de una lista se escribe el nombre de la lista y entre paréntesis se escribe el n-ésimo elemento que se desea, y así se evita el uso del comando GET.

```
« HEAD(INFORM("VOLUMEN Y SUPERFICIE",
  {{"ALTO:","",0,9},
 {"ANCHO:","",0,9},{"LARGO:","",0,9}},
 {1,0},{ },{ })) → L
« L(1) → AL
« L(2) → AN
« L(3) → LA
« AL*LA*AN → VOL
« AL*AN*2+AL*LA*2+AN*LA*2 → SUP
« {"VOLUMEN: "+VOL,"SUPERFICIE:"+SUP}
»
»
»
»
»
»
» ►VOLySUP
```

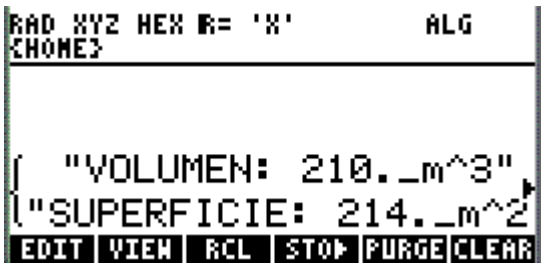
Ejemplo: Ídem, pero haciendo los cálculos dentro de la lista. Son necesarios los paréntesis cuando hay sumas y restas.

```
« HEAD(INFORM("VOLUMEN Y SUPERFICIE",
  {{"ALTO:","",0,9},
  {"ANCHO:","",0,9},{"LARGO:","",0,9}},
  {1,0},{ },{ })) → L
« L(1) → AL
« L(2) → AN
« L(3) → LA
« {"VOLUMEN:"+AL*LA*AN,
   "SUPERFICIE:"+2*(AL*AN+AL*LA+AN*LA)}
»
»
»
»
» ►VOLySUP
```

Ejemplo: Ídem, pero simplificando el algoritmo. Dado que sólo hay tres variables no hay gran complejidad, pero si hay más variables no sería aconsejable reducir tanto el algoritmo porque se dificulta la detección de errores. Para una mejor visualización de la respuesta, se ha seleccionado el indicador de sistema -97 (ver listas en forma vertical).

```
« HEAD(INFORM("VOLUMEN Y SUPERFICIE",
  {{"ALTO:","",0,9},
```

```
{"ANCHO:","EN METROS",0,9},{"LARGO:","",0,9}},
{1,0},{ },{ })) → L
« {"VOLUMEN:"+L(1)*L(2)*L(3)+"_m^3","SUPERFICIE: "
+2*(L(1)*L(2)+L(1)*L(3)+L(2)*L(3))+"_m^2"}
»
» ▶VOLySUP
```



A continuación se presentan algunos comandos de salida.

TEXT

Sirve para escribir texto. La sintaxis es: TEXT("Texto").
No es un comando muy útil.

CLLCD

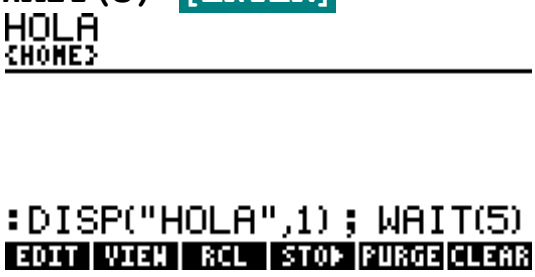
Sirve para limpiar la pantalla. No tiene argumentos de entrada ni de salida. La pantalla queda en blanco hasta que se presione alguna tecla.

DISP

Sirve para desplegar en pantalla momentáneamente lo que se desea en donde se desea. Se combina con WAIT si se quiere que dure más tiempo. La sintaxis es:
DISP("Texto",Número de línea donde se desea ver). No tiene argumentos de salida.
En la HP49G hay siete líneas. La número 1 es la superior y la 7, la inferior.

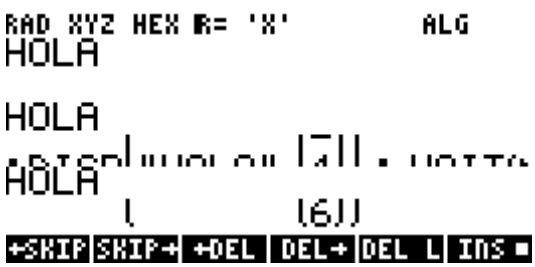
Ejemplo: Se ve "HOLA" en la esquina superior izquierda durante 5 segundos.

```
DISP("HOLA",1),WAIT(5)  [ENTER]
```



Ejemplo: Se ve "HOLA" en las líneas pares.

```
DISP("HOLA",{2,4,6}),WAIT(5)  [ENTER]
```

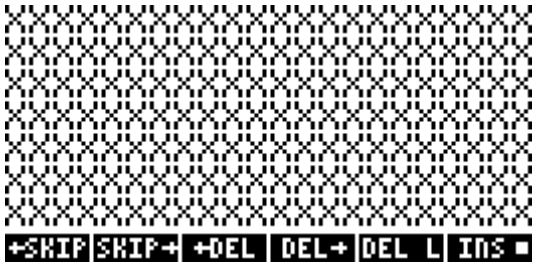


Ejemplo: Ídem, pero limpiando primero la pantalla.

CLLCD,DISP("HOLA",{2,4,6}),WAIT(5) **[ENTER]**

Ejemplo: Fondo. Se escribe cualquier letra o símbolo 22 veces.

DISP("XXXXXXXXXXXXXXXXXXXXXX",{1,2,3,4,5,6,7}),
WAIT(5) **[ENTER]**



También es posible escribir cosas diferentes en líneas diferentes.

Ejemplo:

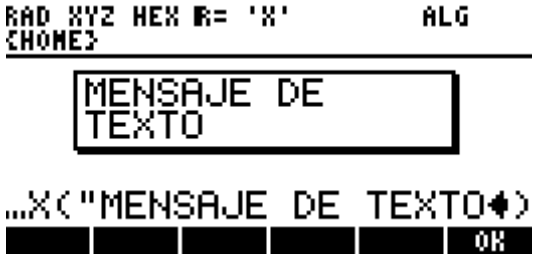
DISP({"X","XX","XXX","XXXX","XXXXX","XXXXXX","XXXXXXX
"},{1,2,3,4,5,6,7}); WAIT(5) **[ENTER]**

MSGBOX

Sirve para hacer cuadros de texto que parecen estar sobre la pantalla. La sintaxis es: MSGBOX("Texto"). El mensaje aparece por tiempo indefinido hasta que se presione ON, ENTER o F6. Sólo se pueden escribir 14 caracteres en una línea.

Ejemplo:

MSGBOX("MENSAJE DE TEXTO") **[ENTER]**

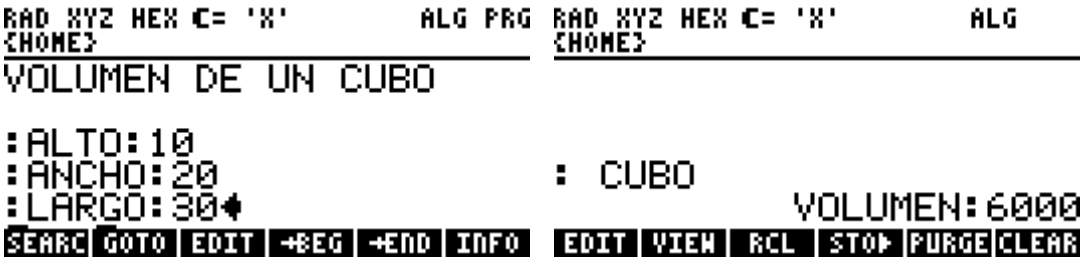


→TAG

Sirve para etiquetar objetos. Sintaxis: →TAG(Objeto,"Nombre"). La ventaja de usar un número etiquetado es que se puede operar posteriormente, mientras que no es posible hacerlo con una cadena.

Ejemplo: Programa que calcula el volumen de un cubo.

```
« →TAG(∏LIST(OBJ→(INPUT("VOLUMEN DE UN CUBO",
  {":ALTO: ↵
  :ANCHO: ↵
  :LARGO:",{1,0}})))," VOLUMEN")
» ▶CUBO
```



Si se necesita sumar este valor (6000) con 500 sólo se presiona +500 **[ENTER]**. La respuesta sería 6500 (sin etiqueta).

COMPARACIONES

Los operadores lógicos sirven para hacer comparaciones. Una expresión es verdadera si su valor es diferente de 0, y es falsa sólo si su valor es 0. Los operadores lógicos son AND, OR XOR y NOT.

AND

Su resultado es verdadero (1 lógico) si todas las expresiones que se comparan son verdaderas. En caso contrario el resultado es falso (0 lógico).

Ejemplo:

5 AND 3 da como resultado 1.
2 AND -3 AND 1 da como resultado 1.
0 AND 7 da como resultado 0.

OR

Su resultado es verdadero si al menos una de las expresiones que se compara es verdadera. Su resultado es falso únicamente si todas las expresiones son falsas.

Ejemplo:

5 OR 3 da como resultado 1.
2 OR -3 OR 1 da como resultado 1.
0 OR 7 OR 0 da como resultado 1.
0 OR 0 da como resultado 0.

XOR

Su resultado es falso si las expresiones son iguales, y es verdadero si las expresiones son diferentes.

Ejemplo:

1 XOR 3 da como resultado 0.

1 XOR -1 da como resultado 0.
0 XOR 7 da como resultado 1.
0 XOR 0 da como resultado 0.

NOT
Cambia el valor lógico de un valor.
Ejemplo:
NOT 0 da como resultado 1.
NOT 1 da como resultado 0.
NOT -32 da como resultado 0.

SAME
Compara dos expresiones, pero es más riguroso que = =.
Ejemplo:
1 SAME 1 da como resultado 1.
1 SAME 5 da como resultado 0.
1 SAME 1.0 da como resultado 0.

TYPE
Su resultado es el tipo de objeto al que pertenece el argumento. Algunos tipos son presentados en la página 7.
Ejemplo:
TYPE(3) da como resultado 28 (número entero).
TYPE(X) da como resultado 6 (variable global).
TYPE(1.35) da como resultado 0 (número real).

SF
Establece un indicador del sistema (flag). Si se verifica en [MODE] [FLAGS] el indicador de sistema deseado aparece con un cheque.
Ejemplo:
SF(-97) pone un cheque al indicador de sistema -97, haciendo que las listas se vean en forma vertical.

CF
Hace lo contrario que SF.
Ejemplo:
CF(-97) quita un cheque al indicador de sistema -97, haciendo que las listas se vean en forma horizontal.

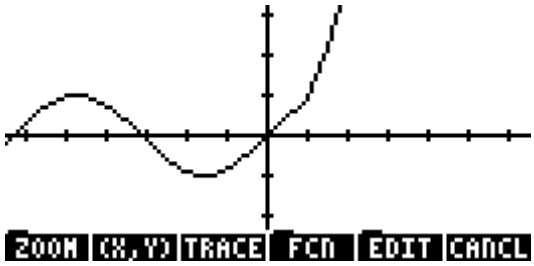
FS? y FC?
Verifican si un indicador de sistema ha sido establecido o no, es decir que verifican si tienen cheque (FS?) o no tiene cheque (CF?).

BIFURCACIONES E ITERACIONES

IF – THEN – ELSE
Sirve para tomar decisiones. Sintaxis: IF Condición THEN Proceso ELSE Proceso Alternativo END. Si se cumple la condición, se ejecuta el proceso de THEN, de lo contrario se ejecuta el de ELSE.

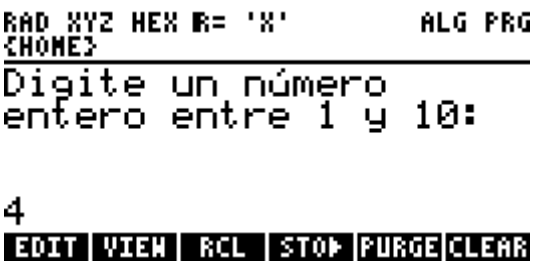
Ejemplo: Graficar x^2 si x>1, de lo contrario graficar seno(x).

```
« IF X >1 THEN X^2 ELSE SIN(X) END »
```



Ejemplo: Juego de adivinanza. RAND produce un número real aleatorio mayor que cero y menor que 1. CEIL redondea un número real al entero mayor más próximo.
Primero se introduce un valor (que se llamará X), el cual es comparado con el número secreto aleatorio de la calculadora (que se llamará XX), y aparece un mensaje que dice si son iguales o diferentes y muestra el número secreto.

```
« OBJ→(INPUT("Digite un número ↵
  entero entre 1 y 10:","")) → X
« CEIL(10*RAND) → XX
«
  IF X == XX
  THEN
  DISP("XXXXXXXXXXXXXXXXXXXXXX",{1,2,3,4,5,6,7}),
  WAIT(1),MSGBOX("GANASTE: "+XX)
  ELSE
  DISP("▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒",{1,2,3,4,5,6,7}),
  WAIT(1),MSGBOX("PERDISTE: "+XX)
  END
»
»
» ▶ADIVINAR
```



Ejemplo: Programa que verifica si un número es impar negativo. MOD obtiene el residuo de la división de X entre 2. Si el residuo es 0, significa que X es par. Para que un número sea impar negativo debe cumplir dos condiciones: que sea menor que cero y que sea impar (en otras palabras, que no sea par)

```
« → X
« IF X<0 AND X MOD 2 ≠ 0
  THEN X+" ES IMPAR NEGATIVO"
  ELSE X+" NO ES IMPAR NEGATIVO"
  END
»
```

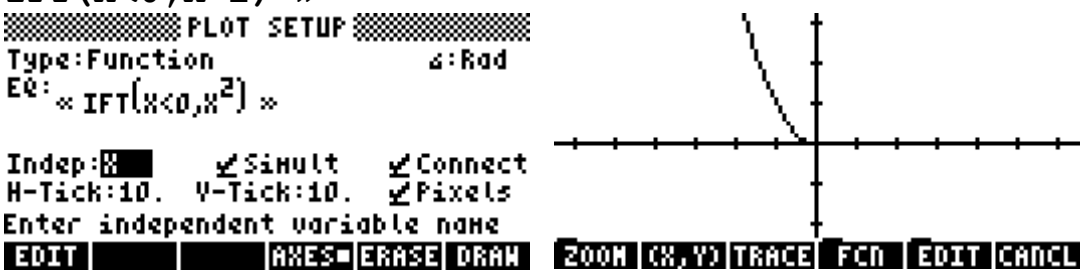
» ▶ **IMPNEG**

IFT e IFTE

Hacen exactamente lo mismo que IF – THEN – ELSE. Sintaxis:
IFT(Condición, Proceso)
IFTE(Condición, Proceso, Proceso Alternativo)

Ejemplo: Si x<0 se grafica x^2. De lo contrario no hay gráfica.

```
« IFT(X<0,X^2) »
```



Los siguientes comandos sirven para hacer iteraciones.

FOR – NEXT

Sintaxis: FOR(Variable, Inicio, Fin) Proceso NEXT.
La variable va aumentando de uno en uno y se ejecuta el proceso (o procesos) hasta que la variable (a veces llamada *bandera* o *contador*) toma el último valor especificado.
La salida es una lista conteniendo los resultados de la iteración.

Ejemplo: Elevar desde 1 hasta 5 a la cuarta potencia.

FOR (X,1,5) X^4 NEXT [ENTER]

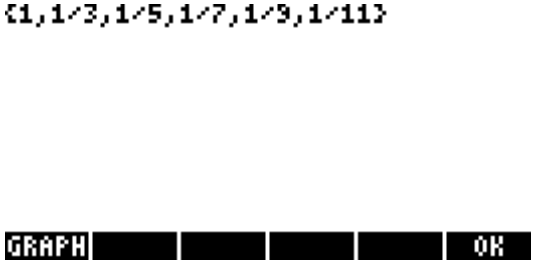
FOR – STEP

Hace lo mismo que FOR – NEXT, pero la variable puede aumentar o decrecer según convenga.
Sintaxis: FOR(Variable, Inicio, Fin,) Proceso STEP(Incremento o decremento).

Ejemplo: Obtener los inversos de los números desde 1 hasta 12, aumentando de dos en dos.

FOR (R,1,12) INV(R) STEP(2) [ENTER]

[TOOL] [F2]



Al ir aumentando de dos en dos, el ultimo número menor o igual que 12 es 11.

Ejemplo: El incremento o decremento puede ser una expresión matemática.

FOR (R,1,12) INV(R) STEP(2+R) [ENTER]

[TOOL] [F2]

{1,1/4,1/10}

GRAPH OK

START – NEXT y START – STEP

START hace lo mismo que FOR, pero la variable siempre es X (o la variable en uso actual). Dicha variable no puede ser usada en el código de proceso.

Sintaxis:

START (Inicio, Fin) Proceso NEXT

START (Inicio, Fin) Proceso STEP (Incremento o decremento)

Ejemplo: La variable no es reconocida dentro del proceso.

START (1,12) INV(X) STEP(3) [ENTER]

[TOOL] [F2]

{1/X,1/X,1/X,1/X}

GRAPH OK

Ejemplo: Obtener 4 números reales aleatorios positivos menores que 1.

START(2,5) RAND NEXT [ENTER]

[TOOL] [F2]

[.156083179654]
[.561556952003]
[.315753873725]
[.722319935785]

TEXT OK

Si el inicio es 2 y el final es 5, hay 4 números en ese intervalo. Es como si el inicio fuera 1 y el final fuera 4.

DO – UNTIL

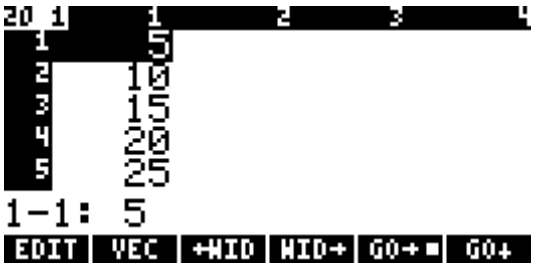
Se ejecuta el proceso *hasta* que se cumpla una condición. El proceso se ejecuta al menos una vez. En otras palabras, la iteración deja de repetirse cuando se cumpla la condición. La sintaxis es la siguiente:

DO Proceso UNTIL Condición END.

Ejemplo: Obtener los múltiplos de 5 menores o iguales que 100. Presentar los resultados en una matriz.
Primero la variable Z (bandera o contador) valdrá 5. Luego, en la siguiente iteración, valdrá 10, y en las siguientes iteraciones irá aumentando su valor *hasta* llegar a 100.

```
« { }▶M; 5 → Z
« DO M+Z▶M; Z+5▶Z; UNTIL Z-5 ≥100 END;
  COL→(AXL(M),1)
»
»▶MULTIPLO
```

MULTIPLO **[ENTER]** **[▼]**



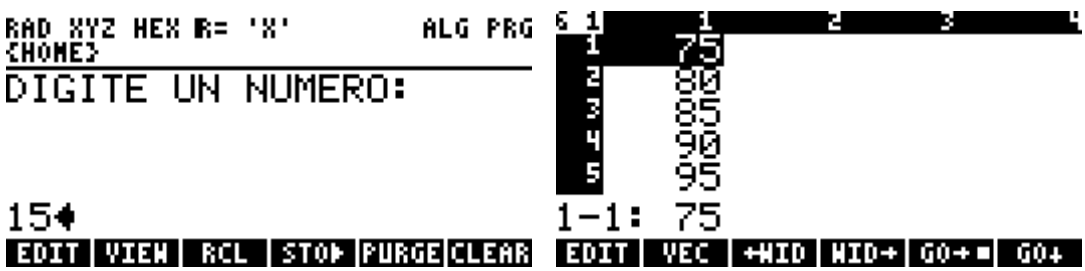
{ }▶M crea una lista vacía con el nombre de M. M+Z agrega a M el número Z. No es lo mismo M+Z que Z+M. Si M fuera la lista {1, 2, 3} y Z fuera 4, M+Z produciría {1, 2, 3, 4}, y Z+M produciría {4, 1, 2, 3}.
Es importante hacer crecer o decrecer el valor del contador dentro del proceso. De lo contrario podría tenerse una iteración infinita.

WHILE – REPEAT

Similar a DO – UNTIL, pero primero se evalúa la condición, y si no se cumple, entonces el proceso puede no ejecutarse. El proceso se ejecutará *mientras* la condición se cumpla.
Sintaxis: WHILE Condición REPEAT Proceso END.

Ejemplo: Se obtienen los múltiplos de 5 menores o iguales a 100 a partir del número introducido.

```
« OBJ→(INPUT("Digite un número","")) → Z
« { }▶M;
  WHILE 5Z ≤100 REPEAT M+5Z▶M; Z+1▶Z; END;
  COL→(AXL(M),1)
»
»▶MULTIPLO
```



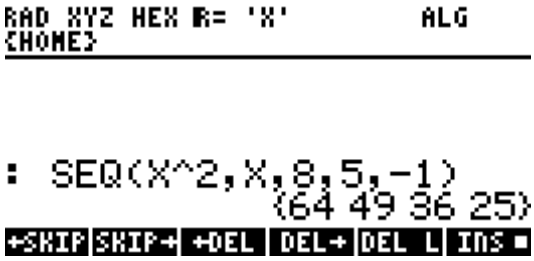
CASE

Por razones desconocidas para el autor, CASE ***no*** está disponible en modo algebraico.

SEQ

Similar a FOR – STEP, pero más simple.
Sintaxis: SEQ(Proceso, Variable, Inicio, Fin, Incremento o decremento)
Ejemplo: Obtener los cuadrados en orden descendente del 5 hasta el 8.

```
SEQ(X^2,X,8,5,-1)
```



EDICIÓN DE PROGRAMAS

Es preferible que se trabaje en modo exacto. Si la calculadora está en modo aproximado a cada número entero lo convierte en real, aumentando el tamaño del programa casi al doble.
Si se pretende hacer grandes cambios al programa es bueno hacer una copia de seguridad.
Si la respuesta del programa no está simplificada (es mostrada como una expresión) puede usarse el comando EVAL o el comando SIMPLIFY, el cual se encuentra disponible en la versión 1.19 del sistema operativo. SIMPLIFY simplifica una expresión mejor que EVAL.

PROGRAMAS MISCELÁNEOS

MATEMÁTICA

Ecuación punto-pendiente. Es útil cuando uno menos se lo espera. La ecuación es:

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0} * (x - x_0)$$

```
« HEAD(INFORM("PUNTO-PENDIENTE",
  {{"X0:","",0,9},{"Y0:","",0,9},{"X1:","",0,9},{"Y0:"
  ,"",0,9}},{1,0},{ },{0,0,0,0})) → M
« EVAL(M(2)+(M(4)-M(2))/(M(3)-M(1))*(X-M(1)))
»
»  ▶PuntoPendiente
```

La respuesta puede tener números reales muy largos. Para leer más fácil la respuesta se puede agregar los comando ENG (aproximar usando múltiplos de 1000), FIX (aproximar usando los decimales deseados), o SCI (notación científica).
`ENG(3);EVAL(M(2)+(M(4)-M(2))/(M(3)-M(1))*(X-M(1)))` produciría que la respuesta sea aproximada usando múltiplos de 1000 y mostrando 3 cifras significativas.
Es importante que la variable X no tenga algún valor asignado.

MÉTODOS NUMÉRICOS o ANÁLISIS NUMÉRICO

Método de la bisección mostrando todas las respuestas. Las respuestas se presentan en una tabla, así:

A	B	F_A*F_P (+/-)	P	Error

```
« HEAD(INFORM("BISECCION",
```

```
   {{"A:", "",0,9},{"B:", "",0,9},
   {"ε:","Error",0},{"F(X):","",9,6}},
   {1,0},{ },{ })) → L
« L(1) → A1
« L(2) → B1
« L(3) → ε
« L(4) → F
«
A1+(B1-A1)/2▶P0;
→NUM(F|(X=A1))▶FA;
→NUM(F|(X=P0))▶FP0;
{A1}▶a; {B1}▶b;
{SIGN(FA*FP0)}▶S;
{P0}▶p; {P0}▶E;
DO
   IF FA*FP0<0
   THEN a+A1▶a; P0▶B1; b+B1▶b; A1+(B1-A1)/2▶P1;
        p+P1▶p;
   ELSE b+B1▶b; P0▶A1; a+A1▶a; A1+(B1-A1)/2▶P1;
        p+P1▶p;
   END;
 ABS(P1-P0)▶ER; A1+(B1-A)/2▶P0;
 →NUM(F|(X=A1))▶FA; →NUM(F|(X=P0))▶FP0;
 S+SIGN(FA*FP0)▶S; E+ER▶E;
UNTIL ε >ER END;
COL→(AXL(A+a),AXL(B+b),AXL(Fa*Fp+S),AXL(P+p),
AXL(Error+E),5)▶R; PURGE({'a', 'b', 'S', 'p', 'E',
'ER', 'P0', 'FA', 'FP0', 'P1'}); R
»
»
»
»
»
»▶BISECCION
```

Aunque en HP User RPL hay diferencia entre letras mayúsculas y minúsculas, se ha nombrado *A1* y *B1* a las primeras variables dado que más adelante se usan las variables *A* y *B* para hacer la tabla de respuesta. (Ejercicio: Probar si funciona el programa nombrando las variables A1 y B1 simplemente como A y B). Es necesario que las variables *A, B, Fa, Fp* (con *p* minúscula; FP es un comando que sirve quién sabe para qué), *P* y *Error* no tengan asignado algún valor. La razón porque se usan variables simbólicas en la tabla de respuesta y no cadenas de texto es que la tabla de respuesta es una matriz, y las matrices y vectores (o arreglos) no permiten que se escriban cadenas de texto, sino sólo números o variables simbólicas (π es una variable numérica y sí puede ser escrita en una matriz o arreglo).
La barra vertical (|) es el comando WHERE. Se obtiene presionando [R-Shift] [I]. Se usa para sustituir nombres o evaluar valores en una expresión, similar al comando SUBST. Su sintaxis es: Función|(X=Valor), donde Valor puede ser un número o una expresión.

2X+COS(X)|(X=5X1) produciría 2*5*X1+COS(5*X1); mientras que 3X|(X=10)+X^2|(X=5) produciría 3*10+5^2 (sin simplificar).

PURGE sirve para borrar variables globales innecesarias. A1, B1, ε y F no fueron borradas porque eran variables locales, aunque A1 y B1 también fueron guardadas como variables globales.

En INFORM se usa el tipo ***nombre global*** (`{"F(X):","",9,6}`) para la función a evaluar porque X es considerada por la calculadora como una variable global y no como un ***objeto algebraico***. En cambio 2X, COS(X), X-5, LN(X), etc., son objetos algebraicos.

Ejercicio: Hacer el programa anterior utilizando variables globales en vez de variables locales para evitar hacer anidamientos. Llamarlo BISECCION2. Después ir al administrador de archivos (**FILES**). ¿Cuál de los dos programas usa más bytes? Luego hacer que la calculadora opere en modo aproximado, volver al administrador de archivos y editar cualquiera de los programas, no hacer ningún cambio y presionar [ENTER]. ¿Cuántos bytes ocupa ahora el programa? Editar nuevamente el mismo programa. ¿Qué cambió en los números?

Método de la bisección. Sólo muestra la respuesta final.

```
« HEAD(INFORM("BISECCION",
  {{"A:", "",0,9},{"B:", "",0,9},
  {"ε:","Error",0},{"F(X):","",9,6}},
  {1,0},{ },{ })) → L
« L(1) → A
« L(2) → B
« L(3) → ε
« L(4) → F
« A+(B-A)/2▶P0; →NUM(F|(X=A))▶FA; →NUM(F|(X=P0))▶FP0;
DO
   IF FA*FP0<0 THEN P0▶B; A+(B-A)/2▶P;
   ELSE P0▶A; A+(B-A)/2▶P;
   END;
   ABS(P-P0)▶ER; A+(B-A)/2▶P0; →NUM(F|(X=A))▶FA;
   →NUM(F|(X=P0))▶FP0;
UNTIL ε >ER END; PURGE({'P0', 'FA', 'FP0', 'ER'}); P
»
»
»
»
»
»▶BISECCION
```

Método de punto fijo. Muestra todas las respuestas en una tabla, así:

P_0	P=G(P)	Error

```
« HEAD(INFORM("PUNTO FIJO",
  {{"P0:", "",0,9},{"G(X):", "función de X",9,6},
  {"ε:","Error",0}},{1,0},{ },{ })) → L
« L(1) → P0
```

```
« L(2) → G
« L(3) → ε
« →NUM(G|(X=P0))▶P; {P0}▶p0; {P}▶p; {ABS(P-P0)}▶E;
DO
  ABS(P-P0)▶ER;P▶P0; →NUM(G|(X=P0))▶P;
  p0+P0▶p0; p+P▶p; E+ER▶E;
UNTIL ε>ER END;
COL→(AXL(p0),AXL(p),AXL(E),3)▶R; PURGE({ 'p', 'p0',
'E', 'ER', 'P0', 'P'}); MSGBOX("RESPUESTA:
[P0,G(P)=P,ERROR]");R
»
»
»
»
»▶PuntoFijo
```

Método de punto fijo. Sólo muestra la respuesta final.

```
« HEAD(INFORM("PUNTO FIJO",
  {{"P0:", "",0,9}, {"G(X):", "función de X",9,6},
  {"ε:","Error",0}},{1,0},{ },{ })) → L
« L(1) → P0
« L(2) → G
« L(3) → ε
« →NUM(G|(X=P0))▶P;
DO
  ABS(P-P0)▶E;P▶P0; →NUM(G|(X=P0))▶P;
UNTIL ε>E END;
PURGE('E');P
»
»
»
»
»▶PuntoFijo
```

Método de Newton-Raphson. La respuesta es mostrada en una tabla, así:

P_0	$F(P_0)$	$F'(P_0)$	P	Error

```
« HEAD(INFORM("NEWTON-RAPHSON",{{"P0:","",0,9},
  {"F(X):","",6,9},{"ε:","",0}},{1,0},{ },{ })) → L
« L(1) → P0
« L(2) → F
« L(3) → ε
« DERVX(F) → D
« { }▶p0; { }▶p; { }▶f; { }▶d; { }▶E;
  DO →NUM(F|(X=P0))▶F0; →NUM(D|(X=P0))▶D0;
  P0-F0/D0▶P; ABS(P-P0)▶ER; p0+P0▶p0; p+P▶p; f+F0▶f;
  d+D0▶d; E+ER▶E; P▶P0;
```

```
  UNTIL ε >ER END;
  COL→(AXL(p0),AXL(f),AXL(d),AXL(p),AXL(E),5)▶R;
  PURGE({'p0','p','P','f','d','E'}); R
»
»
»
»
»
» ▶NewtonRaphson
```

Método de Newton-Raphson. Sólo muestra la respuesta final.

```
« HEAD(INFORM("NEWTON-RAPHSON",{{"P0:","",0,9},
  {"F(X):","",6,9},{"ε:","",0}},{1,0},{ },{ })) → L
« L(1) → P0
« L(2) → F
« L(3) → ε
« DERVX(F) → D
« DO →NUM(F|(X=P0))▶F0; →NUM(D|(X=P0))▶D0;
  P0-F0/D0▶P; ABS(P-P0)▶ER; P▶P0;
  UNTIL ε >ER END; PURGE({'F0','D0','ER'}); P
»
»
»
»
»
» ▶NewtonRaphson
```

ELECTRO

Paralelo. Los valores pueden ser reales o complejos. Los valores son introducidos en una lista. No hay límite de valores de entrada. La salida es un número o una expresión algebraica.

```
« → X 'EVAL(INV(ΣLIST(INV(X))))'
» ▶PARALELO
```

Circuito R-L-C. Puede ser serie o paralelo. Proporciona los valores de α y ω_0. Dichos valores quedan almacenados en variables globales para posterior uso.

```
« HEAD(INFORM("R-L-C",
   {{"R:","",0},{"L:","",0},{"C:","",0}},
  {1,1},{ },{ })) → M
« M(1) → R
« M(2) → L
« M(3) → C
```

```
« 1/√(L*C)▶ω0;
   HEAD(OBJ→(HEAD(CHOOSE("SERIE/PARALELO",
   {"1.SERIE","2.PARALELO"},1)))) → SP
« IF SP==2 THEN 1/(2*R*C)▶α;
   ELSE IF SP==1 THEN R/(2*L)▶α;
    END
   END; {→TAG(α,"α"), →TAG(ω0,"ω0")}
»
»
»
»
»
»▶RLC
```

Series de Fourier. Proporciona los valores de los coeficientes a_0, a_n y b_n. La función periódica puede ser seccionada.

```
« CF(-105); HEAD(INFORM("SERIES DE FOURIER",
  {{"PERIODO:","",0,9},{"No. DE TRAMOS:", "",0}},
  {1,0},{ },{ })) → S
« XQ(S(1))▶T; XQ(S(2)) → TRA
« 0▶a0; 0▶an; 0▶bn;
   FOR(a,1,TRA)
   HEAD(INFORM("SERIES DE FOURIER",
   {{"f(X):","FUNCION DEL TRAMO "+a,0,6,9},
   {"LIM. INFERIOR:","",0,9},
   {"LIM. SUPERIOR:","",0,9}},{1,0},{ },{ })) → L
« XQ(L(1)) → FUN
« XQ(L(2)) → X1
« XQ(L(3)) → X2
« 1/T*∫(X1,X2,FUN,X)+XQ(a0)▶a0;
 2/ T*∫(X1,X2,FUN*COS(2*n*π*X/T),X)+XQ(an)▶an;
 2/ T*∫(X1,X2,FUN*SIN(2*n*π*X/T),X)+XQ(bn)▶bn
»
»
»
» NEXT; SIMPLIFY({→TAG(XQ(a0),"a0"),
  →TAG(XQ(an),"an"),→TAG(XQ(bn),"bn")})
»
»
»▶CoeficientesDeFourier
```

El programa funciona más rápido si la calculadora está en modo exacto, y para asegurarse de lo anterior se usa el comando CF y el indicador del sistema -105.
Durante los cálculos surgen números reales. El comando XQ sirve para convertir números reales en números racionales. Si no está disponible el comando SIMPLIFY se puede remplazar por EVAL.

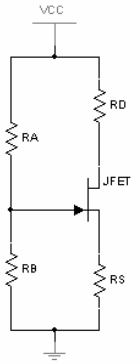
Una forma de comprobar que lo que se ha hecho está correcto es graficar la serie a partir de los coeficientes. Diez términos son suficientes para obtener una gráfica muy buena. La respuesta se guarda en la variable EQ. Sólo hay que presionar [L-Shift] [F4] [F5] [F6] (Plot Setup – Erase – Draw) para obtener la gráfica.

```
« SF(-105);
  a0+Σ(n=1,10,an*COS(2*n*π*X/T)+bn*SIN(2*n*π*X/T))▶EQ
» ▶GraficarSerie
```

SF(-105) sirve para configurar la calculadora en modo aproximado. Así la sumatoria se ejecuta más rápido.
Para acelerar más la gráfica se puede cambiar el paso (step) que la calculadora toma para hacer la evaluación de la función. El paso puede ser 2 ó 3, y se puede decidir si dicho número se refiere a los pixeles o al valor tomado por la variable X. Al graficar la serie la calculadora evalúa uno de cada 2 ó 3 puntos y después los une. Para ello se presiona [L-Shift] [F2] (Plot Window).

JFET de canal N. Para el circuito mostrado proporciona los valores de I_D, V_{GS} y g_m.

```
« HEAD(INFORM("ID Y VGS DE JFET CANAL N",
  {{"VG:","",0},{"VGSOFF:","",0},
  {"IDSS:","",0},{"RS:","",0}},{2,0},{ },{ })) → L
« L(1) → VG
« L(2) → VGSOFF
« L(3) → IDSS
« L(4) → RS
« -(VGSOFF^2*(VGSOFF-2 *RS*IDSS) + √(VGSOFF^6 +
  4*VGSOFF^4*RS*(VG-
  VGSOFF)*IDSS))/(2*RS*VGSOFF*IDSS)▶VGS; (VG-
  VGSOFF)/RS▶ID; ABS(ID/ VGS)▶gm; CLLCD;
  MSGBOX("VGS: "+VGS+"_V ◀┘
  ID: "+ID+"_A ◀┘
  gm: "+gm+"_S")
»
»
»
»
»
»▶JFETN
```



ECONOMÍA

Amortización de un préstamo. Primero se establece el formato de número como FIX con dos decimales, pues se trabaja con dinero. Luego se piden los datos con un cuadro de diálogo. S1 es el capital al inicio del año 1. IN es la tasa de interés. A es el plazo para pagar el préstamo y G representa a los años de gracia.
C es la cuota que se pagará anualmente en los años que nos son de gracia.
La lista S almacena los saldos que se tienen al inicio de cada año.
La lista An almacena los años.
La lista Cu almacena las cuotas que se pagarán en cada año.
La lista Am almacena las amortizaciones que se van haciendo cada año.
La lista Inte almacena los intereses que se pagan anualmente.

Luego se hace un lazo FOR para los años. Para cada año se determina cuánto se pagará de intereses. Luego se toma una decisión: durante los años de gracia no se paga cuota, así que hay que determinar si un año pertenece al periodo de gracia. Si no se paga cuota, entonces no hay amortización al préstamo.
Después se calcula el saldo al inicio del año siguiente, y comienza de nuevo la iteración.

Al terminar el lazo FOR a cada lista se le agrega un título. La lista S almacena el saldo al inicio de un año más que el plazo, lo cual no es útil y se elimina con el comando TAIL, que elimina el primer elemento de una lista. Por ello se invierte el orden de los elementos de S antes de aplicar TAIL con el comando REVLIST, y después se vuelve al orden inicial.

También interesa saber el total de los intereses que se pagarán al final del plazo, y el total de las amortizaciones debe ser igual a la cantidad que se pidió prestada. Se agrega un cero al final de las listas Cu y S porque todas las listas deben tener igual longitud.

Finalmente las listas son convertidas a vectores de columna, y éstos forman una matriz que se mostrará como respuesta.

```
« FIX(2); HEAD(INFORM("AMORTIZACION",
{{"CAPITAL:","DINERO QUE SE PIDE PRESTADO",0},
{"TASA DE INTERÉS:","PORCENTAJE ANUAL, EJ.: 10.5",0},
{"AÑOS DE PLAZO:","INCLUYE PERIODO DE GRACIA",0},
{"AÑOS DE GRACIA:",
"NO SE PAGA CUOTA, SÓLO INTERESES",0}},
{1,1},{},{})) → L
« L(1) → S1
« L(2) → IN
« L(3) → A
« L(4) → G
« S1*(IN/100)*(1+IN/100)^(A-G)/
  ((1+IN/100)^(A-G)-1) → C
« {S1}▶S; {}▶An; {}▶Cu; {}▶Am; {}▶Inte;
  FOR(N,1,A)
    An+N▶An; Inte+S(N)*IN/100▶Inte;
    IF N ≤ G
    THEN
      Cu+0▶Cu; Am+0▶Am
```

```
    ELSE
      Cu+C▶Cu; Am+(C-Inte(N))▶Am
    END;
    S+(S(N)-Am(N))▶S
  NEXT;
AÑO+An+TOTAL▶An;
SaldoInicioAño+REVLIST(TAIL(REVLIST(S)))+0▶S;
CuotaAnual+Cu+0▶Cu;
Intereses+Inte+∑LIST(Inte)▶Inte;
Amortización+Am+∑LIST(Am)▶Am;
COL→(AXL(An),AXL(S),AXL(Cu),AXL(Inte),AXL(Am),5)
»
»
»
»
»
»
»▶AMORTIZACION
```