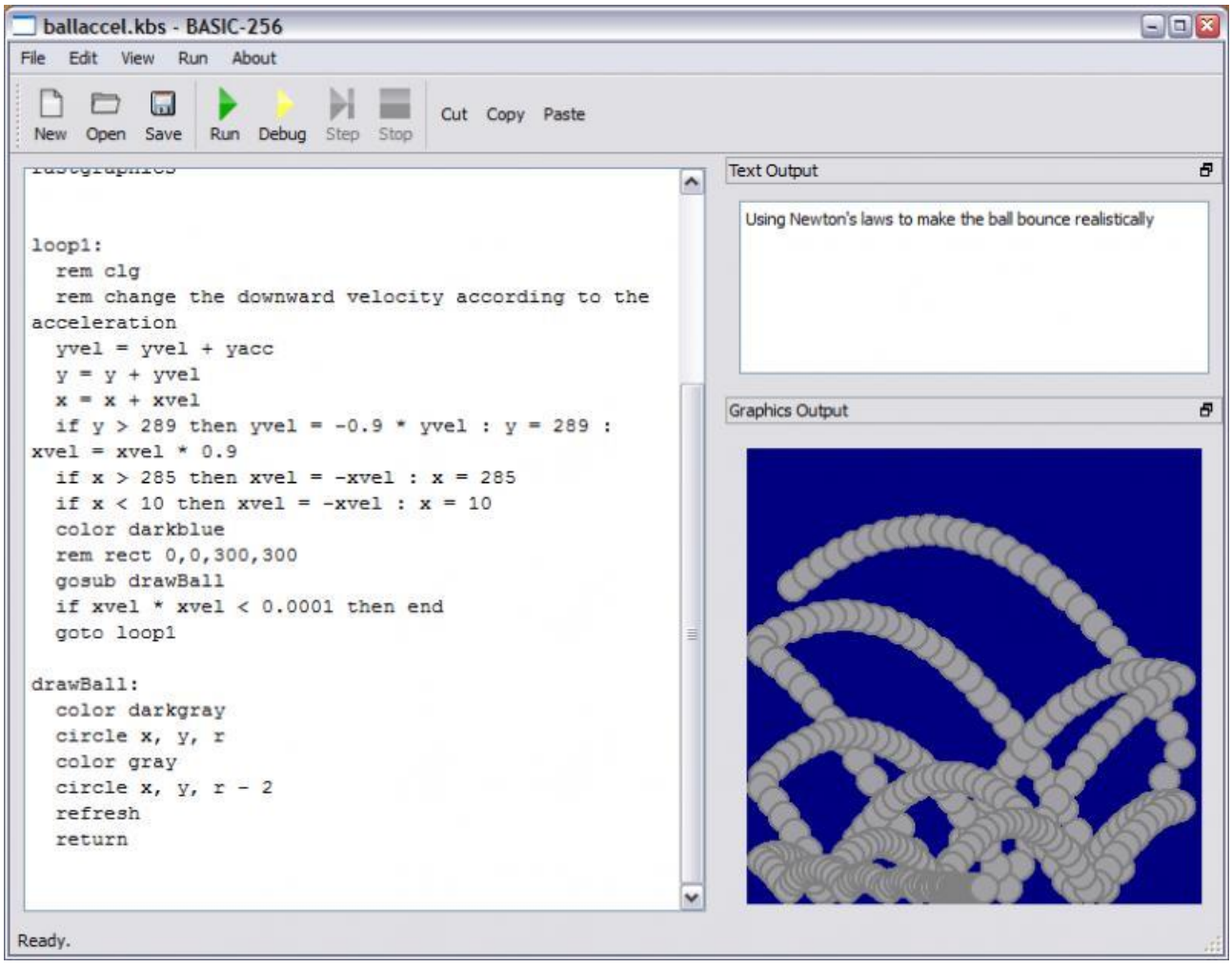
# MANUAL DE BASIC256

# COMANDOS E EXEMPLOS DE PROGRAMAS



# CAPÍTULO 0:
# VARIABLES, ARRAYS e COMENTARIOS

## VARIABLES

Os nomes de variables que conteñen valores numéricos deben comezar cunha letra e poden consistir en calquera número de caracteres alfanuméricos (letras ou números, non espazos nin signos de puntuación). Diferénciase entre maiúsculas e minúsculas.
Os nomes das variables que conteñen cadeas de texto (strings) seguen as mesmas regras que as variables numéricas, pero convén que o seu nome remate co signo dolar ($). Isto xa non é obrigatorio (antes si o era). Pero moito ollo, isto provoca confusións co operador "+" (é a suma das variables numéricas e a concatenación das de texto).
Para asignarlle un valor a unha variable, úsase o operador "=". (Exemplo: a instrución "X = M + 3" almacena na variable X o resultado de sumar o contido actual da variable M máis 3).

## ARRAYS

Os arrays (ás veces chamados vectores) créanse usando o comando DIM. Poden conter datos numéricos ou strings. O acceso a un elemento determinado do array conséguese indicando mediante corchetes a posición do elemento, empezando desde cero. Tamén poden crearse e acceder a arrays de dúas dimensións.

## COMENTARIOS

### INSTRUCIÓN REM

**Formato**
**rem** *comentario*

**Descrición**
Liña de comentario. Unha liña que comeza con REM é ignorada por BASIC256, pero permítenos ir anotando cousas que posteriormente nos aclaren dúbidas sobre o código ou as instrucións.

# CAPÍTULO 1:
# ÁREA DE TEXTO (ENTRADA E SAÍDA)

## INSTRUCIÓN CLS

Borra o contido da ventana de texto.

## INSTRUCIÓN INPUT

**Formato**
**input** *expresión (debe ir entre comillas), VariableCadea*
**input** *expresión (debe ir entre comillas), VariableNumérica*
**input** *VariableCadena*
**input** *VariableNumérica*

**Descrición**
Espera a que o usuario teclee unha liña de texto na área de texto. Cando o usuario pulsa a tecla INTRO ou RETURN, lese a liña e almacénase en *VariableCadea* ou *VariableNumérica*. Opcionalmente, a función pode mostrar o texto *expresión* (que debe ir entre comillas) ao usuario. Tamén poden especificarse referencias aos elementos dun array.

## INSTRUCIÓN PRINT

**Formato**
**print** *expresión* [ ; ]

**Descrición**
Escribe un texto na área de texto, engadindo un salto de liña (ou sexa, o seguinte que escriba será na liña seguinte). Pero se se inclúe o punto e coma opcional, non salta á seguinte liña. Ollo, a instrución PRINT só permite un MÁXIMO DE DOUS elementos. Para crear frases longas, hai que ir poñendo unha instrución PRINT detrás doutra.

EXEMPLO DE PROGRAMA

- Pide un número, almacénao nunha variable e escribe unha mensaxe co seu valor

```
cls
REM A instrución CLS borra a Área de Texto
input "Dame un número ",x
REM A instrución INPUT serve para introducir datos (números ou letras) polo teclado
REM A pregunta aparecerá na Área de Texto. En canto se introduza o dato e se pulse ENTER,
REM ese dato queda almacenado na variable que hai ao final da instrución INPUT.
REM Se queres introducir varios datos diferentes, cada un deles debe ter o seu propio INPUT,
REM e unha variable diferente, xa que en cada variable só pode almacenarse UN dato.
REM A frase entre comiñas do INPUT é opcional, pero normalmente resulta bastante
REM aclaratoria.
print "Metiches o número ";
print x
REM A instrución PRINT serve para amosar mensaxes escritas na Área de Texto.
REM Só pode levar un texto entre comiñas ou unha variable. Se necesitas escribir expresións
REM máis complexas, podes poñer un PRINT detrás doutro, na cantidade que desexes.
REM Se rematas a instrución PRINT cun ";", o seguinte texto escrito con PRINT aparecerá
REM na mesma liña; senón é así, aparecerá na liña seguinte.
```

# CAPÍTULO 2:
# OPERADORES ARITMÉTICOS E MATEMÁTICOS

## OPERADORES ARITMÉTICOS

Os operadores +, -, *, / e ^ úsanse para levar a cabo sumas, restas, multiplicacións, divisións e potencias de números. Os operadores % y \ úsanse para calcular o resto e o cociente da división enteira. Ollo, o operador "+" pode significar suma ou concatenación de strings. (Truco: Se queres sumala, e BASIC256 aínda non ten claro que esa é unha variable numérica, podes multiplicala por 1 e así BASIC256 xa saberá que non é un string, senón un número).

## INSTRUCIÓN ABS

**Formato**
**abs** (*expresión*)

**Descrición**
Devolve o valor absoluto dunha *expresión* numérica.

## INSTRUCIÓN CEIL

**Formato**
**ceil** (*expresión*)

**Descrición**
Devolve o menor enteiro que sexa maior ou igual a *expresión* (aproximanción por exceso)

## INSTRUCIÓN FLOOR

**Formato**
floor (*expresión*)

**Descrición**
Devolve o maior enteiro que sexa menor ou igual que *expresión* (aproximación por defecto).

## INSTRUCIÓN INT

**Formato**
int (*expresión*)

**Descrición**
Converte *expresión* nun número enteiro, truncando os decimais. Dependendo do signo de *expresión*, coincidirá co resultado de CEIL ou FLOOR.

## INSTRUCIÓN RAND

**Formato**
**rand**

**Descrición**

Devolve un número aleatorio entre 0 e 1 (concretamente no intervalo [0, 1)). A distribución dos valores é uniforme.

**Nota**

Para obter números aleatorios noutro intervalo de valores, basta con multiplicar ou sumar os números apropiados, e aplicar a función INT. Por exemplo, para xerar un enteiro entre 0 e 9, pode usarse a expresión int(rand * 10). E para xerar un enteiro entre 1 e 10, pode usarse int(rand * 10) + 1.

## INSTRUCIÓN SQR

**Formato**
**sqr** ( *expresión* )

**Descrición**
Devolve a raíz cadrada de *expresión* (só a positiva).

EXEMPLOS DE PROGRAMAS

- Pide un número decimal e calcula a súa aproximación por exceso

```
cls
input "Dame un número decimal ",x
REM Para introducir números decimais, o caracter separador é o punto "."
print "A súa aproximación por exceso é ";
print ceil (x)
REM As funcións ABS, CEIL, FLOOR, INT, RAND e SQR aplícanse sobre
REM números ou expresións que van entre parénteses despois delas
```

- Xera un número enteiro ao chou entre 1 e 100

```
cls
print "O número entre o 1 e o 100 xerado ao chou é ";
print int (rand * 100) + 1
REM A función RAND xera un número decimal no intervalo (0,1)
REM Se multiplicas ese número por un enteiro, xa podes
REM xerar números decimais que estean no intervalo (0,n).
REM Se lle aplicas a función INT, descartas os decimais
REM e quédaste só coa parte enteira, que é o máis interesante.
REM Se lle sumas ou restas algunha cantidade, podes xerar
REM números enteiros no intervalo que desexes.
```

# CAPÍTULO 3:
# OPERADORES LÓXICOS E COMPARADORES (CONDICIONAL)

## OPERADORES LÓXICOS e COMPARADORES

Os operadores lóxicos son AND (Y,E), OR (O,OU), NOT (NO,NON) e XOR (OR exclusivo).
Os comparadores son <, >, <=, >=, = e <> (distinto).

## INSTRUCIÓN IF-THEN-ELSE

**Formatos**
**if** *ExpresiónCondicional* **then** *instrución*

**if** *ExpresiónCondicional* **then**
*instrución(s)*
**end if**

**if** *ExpresiónCondicional* **then**
*instrución(s)*
**else**
*instrución(s)*
**end if**

**Descrición**
Unha instrución IF de liña única avalía *ExpresiónCondicional*; cando esta sexa verdadeira, a(s) *instrución(s)* se executan, en caso contrario, a execución continúa na liña seguinte. Tamén hai dúas formas de instrución IF multiliña: a primeira cun bloque de instrucións que se executan cando a avaliación da expresión sexa verdadeira; a segunda conta cun bloque de instrucións a executar en caso de resultado verdadeiro e un bloque de instrucións en caso de resultado falso.

EXEMPLOS DE PROGRAMAS

- Pide un número e di se é positivo ou non

```
cls
input "Dime un número ", x
if x > 0 then
REM A instrución IF é a que comproba se unha determinada
REM condición lóxica se cumpre ou non se cumpre.
REM Esa condición, que só pode ser CERTA ou FALSA,
REM ponse entre as palabras IF e THEN
print "O número é positivo"
REM Entre as palabras THEN e ELSE
REM póñense todas as instrucións que queres que se realicen
REM se se cumpre esa condición (poden ser 1 ou moitas).
```

```
else
REM Entre as palabras ELSE e END IF póñense todas as
REM instrucións que queres que se realicen cando a
REM condición non se cumpre (poden ser 1 ou moitas)
print "O número é negativo ou 0"
end if
REM Todo este bloque debe ir rematado con END IF.
REM Pero se a instrución IF-THEN é moi simple, pode ir
REM todo na mesma liña, sen ELSE e sen END IF.
```

# CAPÍTULO 4:
# CONTADORES E ACUMULADORES (BUCLES)

## INSTRUCIÓN FOR-NEXT

**Formato**
**for** *variable = expresión1* **to** *expresión2* [**step** *expresión3*]
*instrución(s)*
**next** *variable*

**Descrición**
As instrucións FOR e NEXT úsanse conxuntamente para executar unha instrución ou grupo de instrucións un número determinado de veces (normalmente coñecido de antemán).
Cando a instrución FOR aparece por primeira vez, a *variable* toma o valor *expresión1*. Despois de cada instrución NEXT, o valor de *variable* aumenta en 1 (por defecto) ou en *expresión3* se se usou a instrución opcional STEP, ata que a *variable* é maior ou igual que *expresión2* (para valores positivos de STEP) ou menor ou igual que *expresión2* (para valores negativos de STEP). *Expresión1*, Expresión2 e *Expresión3* poden ser valores enteiros, decimais ou negativos, pero teñen que ter coherencia aritmética. Ou sexa, se *Expresión1* é menor que *Expresión2*, *Expresión3* debe ser positivo.

## INSTRUCIÓN DO-UNTIL

**Formato**
**do**
*instrución(s)*
**until** *ExpresiónCondicional*

**Descrición**
Executa a(s) *instrución(s)* disposta(s) dentro do bucle **do** mentres a *ExpresiónCondicional* sexa falsa. **Do-Until** executa as instrucións do bucle, como mínimo, unha vez (a diferencia da instrución While-End while, que vén a continuación). A comprobación de *ExpresiónCondicional* faise despois de cada execución do código que está dentro do bucle.

## INSTRUCIÓN WHILE-END WHILE

**Formato**
**while** *ExpresiónCondicional*
*instrución(s)*
**end while**

**Descrición**
Executa a(s) *instrución(s)* dentro do bucle WHILE ata que *ExpresiónCondicional* sexa falsa. O bucle While-End while podería non executar nunca as instrucións do bucle, xa que a avaliación faise antes de que o código que está dentro do bucle sexa executado. Esta é a principal diferencia coa instrución Do-until, da que falamos antes.

## CONTADORES e ACUMULADORES

Os contadores son expresións do tipo:
Variable = Variable + 1
Variable = Variable - 1

Os acumuladores son expresións do tipo:
Variable = Variable + Expresión
Variable = Variable - Expresión

Aínda que dende o punto de vista matemático parecen expresións absurdas, en Informática non o son en absoluto, de feito son moi necesarias en moitos programas. Por exemplo, a instrución "X = X + 1" fai o seguinte: se cando o programa chega a este punto, a variable X valía 3, fai a operación da parte dereita da expresión, e almacena o resultado na variable que está no lado esquerdo. Polo tanto, a partir deste punto, a variable xa valerá 4.
Tendo iso en conta, hai que ter claro que no lado esquerdo dunha destas expresións só pode haber unha variable. Por exemplo, en Informática non terían ningún sentido expresións como "X + 1 = X" ou "3 = X + 2".

EXEMPLOS DE PROGRAMAS

- Escribe os números pares entre o 1 e o 50

```
cls
for n = 2 to 50 step 2
REM Na instrución FOR hai que dicirlle que conxunto de valores
REM tomará secuencialmente (un detrás doutro) unha variable.
REM Coa partícula STEP (que é opcional) podo facer que a
REM variable cambie de valor a maior ritmo ou disminúa.
REM Se non poño STEP, a variable crecerá de 1 en 1.
print n;
print " ";
REM Todas as instrucións que estean entre FOR e NEXT,
REM repetiranse unha vez tras outra, cada vez cun valor
REM diferente da variable que acompañe a FOR e NEXT.
next n
REM Todo este bloque péchase con NEXT, que ten que levar
REM a mesma variable que se puxo no FOR inicial
```

- Pide números por teclado ata que se introduza un 0; ao final di cantos números entraron

```
cls
c = 0
REM "c" é a variable que vai contabilizar a cantidade
REM de números que se introducen ata a entrada do 0 (incluído)
do
REM O bloque que se repite comeza coa instrución DO
REM e remata coa instrución UNTIL
REM Todas as instrucións que hai entre estas dúas palabras
```

```
REM realizaranse, unha vez tras outra, ata que se cumpra
REM a condición lóxica que vai detrás da palabra UNTIL
input "Dime un número (0 para rematar) ",n
c = c + 1
REM Antes de comezar unha nova execución do bloque completo
REM a variable "c" contabiliza un número introducido máis
until n = 0
REM Aquí está a condición que permite a saída do bucle
print "Entraron un total de ";
print c;
print " números"
REM Con este conxunto de PRINTs escribo o resultado final
```

- Pide 5 números por teclado e calcula a suma e produto de todos eles

```
cls
suma = 0
prod = 1
REM "suma" é a variable que vai almacenar a suma total
REM e debe tomar o valor inicial de 0
REM “prod” é a variable que vai almacenar o produto total
REM e debe tomar o valor inicial de 1 (porque senón
REM provocaría que o resultado final fora sempre 0)
For veces = 1 to 5
REM A variable “veces” simplemente fai que o bucle
REM se repita as 5 veces que nós queremos
REM Todas as instrucións que hai entre FOR e NEXT
REM realizaranse, unha vez tras outra, para cada valor de “veces”
input "Dime un número ",n
suma = suma + int(n)
REM Este truco con INT é para que BASIC teña claro que a variable
REM "n" contén números, non letras, non é de tipo "string"
REM Hai que usalo cando o número entra por teclado e se suma
REM Este acumulador vai realizando a suma total
prod = prod * n
REM Este acumulador vai realizando o produto total
Next veces
print "A suma total foi ";
print suma
print "O produto total foi ";
print prod
REM Con este conxunto de PRINTs escribo os resultados finais
```

# CAPÍTULO 5:
# STRINGS (CADEAS DE TEXTO)

## OPERADORES e COMPARADORES

O operador "+" serve para concatenar dous strings ou variables de tipo cadea de texto. Por exemplo, se facemos a operación "Choco" + "late", o resultado será "Chocolate".
Os comparadores lóxicos tamén valen para ordear cadeas de texto ou strings. Por exemplo "A"<"B".
E podemos facer comparacións deste tipo en instrucións como If-then, Do-until ou While-End while.

## INSTRUCIÓN LENGTH

**Formato**
length(*cadea*)

**Descrición**
Devolve o número de caracteres que hai en *cadea*.

## INSTRUCIÓN LEFT

**Formato**
left(*cadea*, *número*)

**Descrición**
Devolve un anaco de *cadea* formada cos primeiros *número* caracteres da cadea (pola esquerda).

## INSTRUCIÓN MID

**Formato**
**mid**(*cadea*, *comezo*, *número*)
**Descrición**
Devolve un anaco da *cadea* especificada, comezando na posición *comezo* e collendo os seguintes *número* caracteres ou ata o final da cadea (o que ocorra antes).

## INSTRUCIÓN RIGHT

**Formato**
**right**(*cadea*, *número*)

**Descrición**
Devolve un anaco de *cadea* formada cos últimos *número* caracteres da cadea (pola dereita).

EXEMPLOS DE PROGRAMAS

- Pide un nome e escribe a súa primeira letra

```
cls
input "Dime o teu nome ",n
print "O teu nome comeza por ";
print left (n, 1)
REM As funcións LEFT, MID e RIGHT serven para manexar textos.
REM Por exemplo, á función LEFT hai que dicirlle cal é o texto
REM (entre comiñas ou unha variable) e o número de caracteres,
REM e collerá ese número de letras comezando pola esquerda.
```

# CAPÍTULO 6:
# ARRAYS (LISTAS E MATRICES)

## INSTRUCIÓN DIM

**Formato**
**dim** *VariableNumérica* ( *NúmeroEntero* )
**dim** *VariableCadena$* ( *NúmeroEntero* )
**dim** *VariableNumérica* ( *filas* , *columnas* )
**dim** *VariableCadena$* ( *filas* , *columnas* )

**Descrición**
Crea e devolve un novo array de lonxitude *NúmeroEntero* ou un array bidimensional organizado en *filas* e *columnas*. Dependendo da variable á que se asigne, créase un array numérico ou de cadeas.

| | |
|---|---|
| datos [0] | Un array (similar a unha lista de Scratch) é un conxunto de variables que comparten nome e que poden almacenar datos relacionados entre sí. Aos datos almacenados en cada elemento da lista accédese poñendo o nome da lista seguido da posición entre corchetes (ollo: comezan a numerar polo 0, non polo 1). Este é un array de 1 dimensión, tamén chamado Lista. |
| datos [1] | |
| datos [2] | |
| datos [3] | |
| datos [4] | |

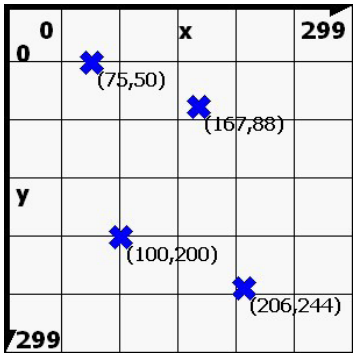
| | | | |
|---|---|---|---|
| datos [0,0] | datos [0,1] | datos [0,2] | Este é un array de 2 dimensións, tamén chamado Matriz. O primeiro índice indica a fila na que está o elemento e o segundo índice indica a columna. Polo tanto, pódense almacenar 12 datos (4x3) |
| datos [1,0] | datos [1,1] | datos [1,2] | |
| datos [2,0] | datos [2,1] | datos [2,2] | |
| datos [3,0] | datos [3,1] | datos [3,2] | |

EXEMPLOS DE PROGRAMAS

- Pide cinco números por teclado, e almacénaos nun array; despois escríbeos na pantalla

```
cls
dim datos (5)
REM Cando queremos crear un array, usamos a instrución DIM
REM e poñemos entre parénteses cantos elementos vai ter.
REM Ollo, numéranse comezando polo 0, así que este array
REM de 5 elementos numerarase do 0 ao 4.
for n = 0 to 4
input "Dime un número ", datos [n]
REM Como vedes, as variables que son arrays, poden ir nas
REM instrucións INPUT, pero o elemento concreto
REM debe especificarse ENTRE CORCHETES (non parénteses).
next n
print "Os números almacenados son: "
for z = 0 to 4
print datos [z];
REM Neste bucle FOR-NEXT escribimos todos os elementos
REM do array na mesma liña pero algo separados entre sí.
print " ";
next z
```

# CAPÍTULO 7:
# ÁREA GRÁFICA (COLOR E DEBUXO)




A área de saída gráfica, por defecto, é de 300 pixeles de ancho (*x*) por 300 pixeles de alto (*y*). Un pixel é o punto máis pequeno que pode ser mostrado no seu monitor. O vértice superior esquerdo é a *orixe* (0,0), e o inferior dereito corresponde a (300,300).

Podes amosar as liñas da grella na área de saída grafica da pantalla marcando a opción Graphics Window Grid Lines no menu View (*Ver*).

## INSTRUCIÓN CLG

Borra o contido da ventana de gráficos.

## INSTRUCIÓN COLOR

**Formato**
**color** *NomedoColor*
**color** (*NomedoColor* )
**color** *valor_rgb*
**color** (*valor_rgb*)

**Descrición**

Establece como color de debuxo actual o *NomedoColor*, o color RGB -composto pola combinación *red*, *green*, *blue*- , ou o valor RGB. Hai un color especial denominado CLEAR, cuxo valor é -1. Se se establece como color actual, os pixels ou figuras que se debuxen borrarán os pixels da área de gráficos e faranos transparentes.

| Nome do color | Valores RGB | |
|---|---|---|
| black | 0, 0, 0 | |
| white | 255, 255, 255 | |
| red | 255, 0, 0 | |
| darkred | 128, 0, 0 | |
| green | 0, 255, 0 | |
| darkgreen | 0, 128, 0 | |
| blue | 0, 0, 255 | |
| darkblue | 0, 0, 128 | |
| cyan | 0, 255, 255 | |
| darkcyan | 0, 128, 128 | |

| | | |
|---|---|---|
| purple | 255, 0, 255 | |
| darkpurple | 128, 0, 128 | |
| yellow | 255, 255, 0 | |
| darkyellow | 128, 128 ,0 | |
| orange | 255, 102, 0 | |
| darkorange | 176, 61 ,0 | |
| grey | 164, 164 ,164 | |
| darkgrey | 128, 128 ,128 | |
| clear | -1 | |

## INSTRUCIÓN RGB

**Formato**
**rgb**( *vermello, verde, azul* )

**Descrición**
Devolve o valor RGB do color definido polas súas compoñentes vermella, verde e azul. Os valores válidos para vermello, verde e azul van do 0 ao 255.

## INSTRUCIÓN CIRCLE

**Formato**

**circle** *x,y,r*

**Descrición**
Debuxa un círculo con centro no punto (x,y) e radio "r", usando a cor actual.

## INSTRUCIÓN LINE

**Formato**

**line** *x1, y1, x2, y2*
**line** (*x1, y1, x2, y2*)

**Descrición**
Traza unha liña dende o punto (x1,y1) ata o punto (x2,y2).

## INSTRUCIÓN PENWIDTH

**Formato**
**penwidth *n***

**Descrición**
Cambia o ancho da "*punta do lápis*" co que se debuxa. A punta do lápis representa o ancho da liña que está sendo debuxada e tamén o ancho da liña a usar para o contorno das formas. O valor "n" indica o número de pixels da anchura do trazo.

## INSTRUCIÓN POLY

**Formato**
**poly** *variable de array numérica*
**poly** {x1, y1, x2, y2, x3, y3 …}

**Descrición**

Debuxa un polígono. Os lados do polígono son definidos polos valores almacenados no array, o cal debe conter pares de puntos (x,y) secuencialmente. A lonxitude do array dividida entre 2 dará o número de puntos total. Un polígono pode tamén especificarse usando unha lista de pares (x,y) entre chaves {}.

## INSTRUCIÓN RECT

**Formato**
**rect** *x,y,anchura,altura*
**rect** (*x, y, anchura, altura*)

**Descrición**
Debuxa un rectángulo de (anchura x altura) pixels, usando a cor actual. A esquina superior esquerda estará situada en (*x, y*).

## INSTRUCIÓN STAMP

**Formato**
**stamp** *x, y, array*
**stamp** *x, y,* {x1, y1, x2, y2, x3, y3 …}
**stamp** *x, y, escala, array*
**stamp** *x, y, escala,* {x1, y1, x2, y2, x3, y3 …}
**stamp** *x, y, escala, rotación, array*
**stamp** *x, y, escala, rotación,* {x1, y1, x2, y2, x3, y3 …}

**Descrición**
Debuxa un polígono coa esquina superior esquerda (orixe) en (x,y). De xeito opcional, o tamaño do polígono pode ser definido por *escala* (1 = tamaño normal). Tamén se pode rotar a imaxe de acordo cun ángulo especificado (en radiáns, no sentido das agullas do reloxo) en torno á orixe. Os lados do polígono son definidos polos valores almacenados no array, o cal debe conter pares (x,y) secuencialmente. A lonxitude do array dividida entre 2 dará o número de puntos. O polígono pode tamén especificarse usando unha lista de pares (x,y) entre chaves {}.

EXEMPLOS DE PROGRAMAS

- Debuxa un rectángulo gris e un círculo azul

```
clg
REM Limpa a Área de Gráficos
color darkgrey
REM Poño como color activo o "darkgrey"
rect 10,10,50,70
REM Pinto un rectángulo coa esquina superior esquerda
REM no punto (10,10), 50 puntos de anchura e 70 de altura
color cyan
REM Poño como color activo o "cyan"
circle 200,200,50
REM Pinto un círculo co centro no punto (200,200)
REM e con 50 puntos de radio
```