
Tortugas y fractales

Las gráficas tortuga fueron añadidas al lenguaje de programación Logo por Seymour Papert a finales de la década de 1960 para apoyar la versión de Papert del robot tortuga, un simple robot controlado desde el puesto de trabajo del usuario diseñado para llevar a cabo funciones de dibujo asignadas mediante una pequeña pluma retractil en su interior o adjuntada al cuerpo del robot.

La tortuga tiene tres atributos:

- Una posición
- Una orientación
- Una pluma, teniendo atributos como color, ancho y arriba y abajo.

La tortuga se mueve con comandos relativos a su posición, como «avanza 10 » y «gira a la izquierda 90 »

Las gráficas tortuga se adaptan bien al dibujo de fractales.

Vamos a explicar cómo construir una tortuga para *Octave* o *Matlab* y luego vamos a desarrollar algún ejemplo clásico de dibujo de fractales.

Lo primero es construir la tortuga. Nuestra tortuga admitirá diferentes comandos identificados cada uno de ellos por una cadena de caracteres. En principio vamos a utilizar dos comandos: 'AVANZA' y 'GIRA'. En ambos casos nos hace falta un valor que defina la magnitud del avance o del giro de la tortuga. La función *turtle()* admitirá, por tanto dos parámetros, quedando su signatura definida de la siguiente manera:

turtle(command, value)

La versión *clásica* de la tortuga de *Papert* admitía dos comandos para el giro: *Gira izquierda* y *Gira derecha*. Nosotros vamos a utilizar un único comando *GIRA*, distinguiendo el sentido de giro mediante el signo del *value*.

La arquitectura de la función *turtle()* estará organizada mediante una función principal, que distribuye los comandos recibidos, y una serie de funciones auxiliares para cada comando. La función principal se encarga de interpretar el comando recibido y pasarlo a la función auxiliar correspondiente que es la que realmente ejecutará el comando. El primer listado de nuestra función *turtle()* podría ser el siguiente:

```
function turtle(command, value)
% Tortuga básica para gráficos

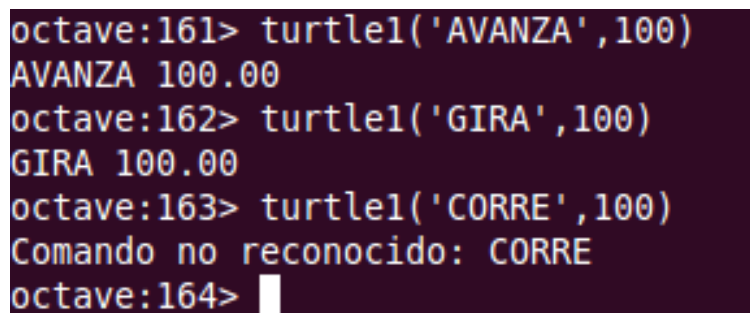
    if strcmp(command, 'GIRA')
        gira(value);
    elseif strcmp(command, 'AVANZA')
        avanza(value);
    else
        dispError(command);
    end
end
```

```
function gira(value)
    fprintf('GIRA %f\n', value)
end

function avanza(value)
    fprintf('AVANZA %f\n', value)
end

function dispError(command)
    fprintf('Comando no reconocido: %s\n', command)
end
```

Podemos probar la función `turtle()` con comandos válidos y no válidos. Por supuesto la tortuga todavía no dibuja nada, lo único que hace es imprimir un mensaje con el comando recibido, de forma que podamos comprobar que la arquitectura *función principal y subfunciones* funciona adecuadamente. Se trata solo del *esqueleto* que nos permitirá ir dotando de funcionalidad a la tortuga.



```
octave:161> turtle1('AVANZA',100)
AVANZA 100.00
octave:162> turtle1('GIRA',100)
GIRA 100.00
octave:163> turtle1('CORRE',100)
Comando no reconocido: CORRE
octave:164> 
```

Inicialmente nuestra tortuga mantendrá los valores correspondientes a la posición y la orientación. La tortuga necesita mantener el valor de sus variables de estatus entre una ejecución y otra. Para ello utilizaremos unas variables globales llamadas `x`, `y` y `angulo`. La función `turtle()` se limita a declarar dichas variables como globales. Será el programa principal que utilice la función `turtle()` el encargado de definir las.

Vamos a añadir dos comando utilitarios adicionales: el comando `'INIT'` y el comando `'DISP'`. El comando `'INIT'` se encargará de inicializar las variables de estatus, poniendo a cero los valores de `x`, `y` y `angulo`. El comando `'DISP'`, por su parte, mostrará en pantalla el valor actual de las variables de estatus de la tortuga.

Con estas nuevas consideraciones, el listado de nuestra tortuga quedará de la siguiente manera:

```
function turtle(command, value)
% Tortuga básica para gráficos
% Necesita la existencia de unas variables globales x, y, angulo

    global x y angulo;

    if strcmp(command, 'GIRA')
        gira(value);
    elseif strcmp(command, 'AVANZA')
        avanza(value);
    elseif strcmp(command, 'DISP')
        disp();
    elseif strcmp(command, 'INIT')
        init();
    else
        dispError(command);
    end
end
```

```
function gira(value)
    fprintf('GIRA %f\n', value)
end

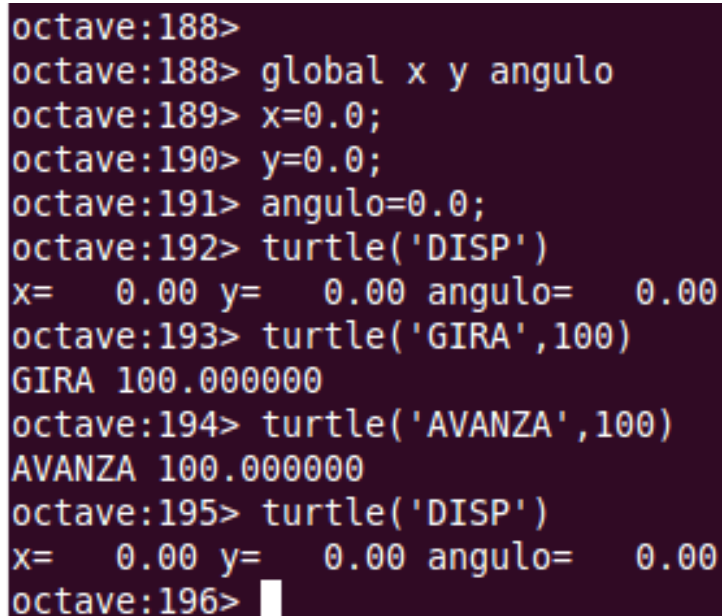
function avanza(value)
    fprintf('AVANZA %f\n', value)
end

function disp()
    global x y angulo;
    fprintf('x= %6.2f y= %6.2f angulo= %6.2f\n', x, y, angulo)
end

function init()
    global x y angulo;
    x=0.0; y=0.0; z=0.0;
    disp();
end

function dispError(command)
    fprintf('Comando no reconocido: %s\n', command)
end
```

Si tratamos de ejecutar la función `turtle()` desde consola nos dará un error, a menos que declaremos como globales y asignemos un valor inicial a las variables `x`, `y` y `angulo`. Podemos ver la secuencia de comandos en la siguiente imagen:



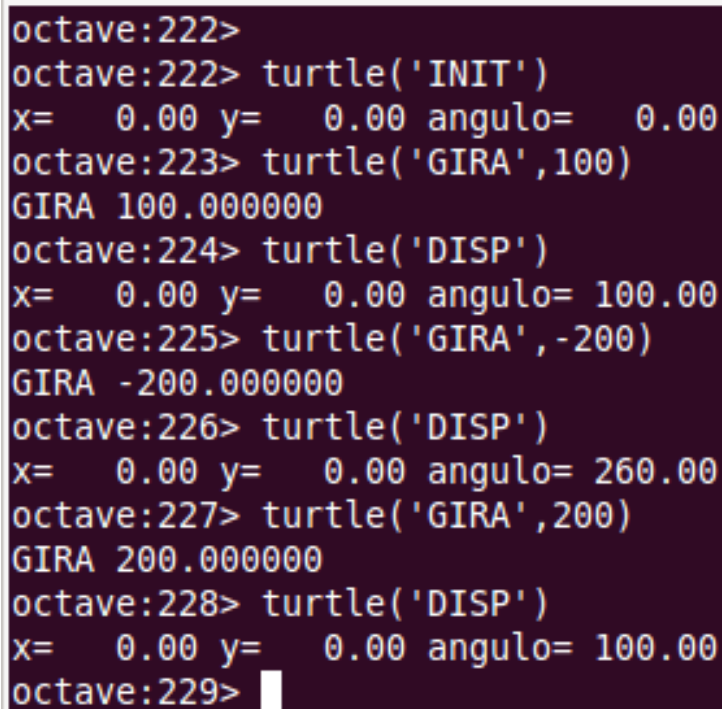
```
octave:188>
octave:188> global x y angulo
octave:189> x=0.0;
octave:190> y=0.0;
octave:191> angulo=0.0;
octave:192> turtle('DISP')
x=  0.00 y=  0.00 angulo=  0.00
octave:193> turtle('GIRA',100)
GIRA 100.000000
octave:194> turtle('AVANZA',100)
AVANZA 100.000000
octave:195> turtle('DISP')
x=  0.00 y=  0.00 angulo=  0.00
octave:196> 
```

Primero definimos desde consola las variables `x`, `y`, `angulo` y luego llamamos a la función `turtle()` varias veces con distintos comandos. La última ejecución del comando `DISP` muestra ceros en las variables de estatus, a pesar de haber realizado un giro y un avance. Esto es debido a que aún no hemos implementado las rutinas correspondientes. Lo que si podemos comprobar es el funcionamiento correcto del mecanismo de variables globales para `x`, `y`, `angulo`.

Vamos a implementar la función `gira()`. Esta función debe actualizar el valor de la variable global `x`, sumándole el valor recibido. Habrá que ajustar el nuevo valor del `angulo` al intervalo `0-360`. El listado de la función 'GIRA' quedará de la siguiente forma:

```
function gira(value)
    global angulo;
    fprintf('GIRA %f\n', value)
    angulo = angulo + value;
    if abs(angulo) > 360
        angulo = rem(angulo, 360);
    end
    if angulo == 360 | angulo == -360
        angulo = 0;
    end
    if angulo < 0
        angulo = 360 + angulo;
    end
end
```

Podemos probar la nueva implementación de la función *gira()* con unas cuantas sentencias desde la consola de *Octave*. (Tendremos que haber realizado anteriormente la declaración y definición de las variables globales *x*, *y*, *angulo*). El resultado se ve en la siguiente figura:



```
octave:222>
octave:222> turtle('INIT')
x=  0.00 y=  0.00 angulo=  0.00
octave:223> turtle('GIRA',100)
GIRA 100.000000
octave:224> turtle('DISP')
x=  0.00 y=  0.00 angulo= 100.00
octave:225> turtle('GIRA',-200)
GIRA -200.000000
octave:226> turtle('DISP')
x=  0.00 y=  0.00 angulo= 260.00
octave:227> turtle('GIRA',200)
GIRA 200.000000
octave:228> turtle('DISP')
x=  0.00 y=  0.00 angulo= 100.00
octave:229> 
```

La implementación de la función *avanza()* requiere algunos cálculos trigonométricos para calcular los incrementos en *x* y *y*. El listado queda de la siguiente manera:

```
function avanza(value)
    global x y angulo;
    fprintf('AVANZA %f\n', value)
    incx = value * cosd(angulo);
    incy = value * sind(angulo);
    x = x + incx;
    y = y + incy;
end
```

Podemos volver a probar la tortuga con comandos 'AVANZA'. En la figura siguiente se puede comprobar el resultado.

Obsérvese que los ángulos de la tortuga se miden desde el eje x en sentido levógiro. La dirección positiva del eje y corresponde al ángulo 90 .

```
octave:253> turtle('INIT')
x=  0.00 y=  0.00 angulo=  0.00
octave:254> turtle('GIRA',90)
GIRA 90.000000
octave:255> turtle('AVANZA',100)
AVANZA 100.000000
octave:256> turtle('DISP')
x=  0.00 y= 100.00 angulo=  90.00
octave:257> turtle('GIRA',90)
GIRA 90.000000
octave:258> turtle('AVANZA',100)
AVANZA 100.000000
octave:259> turtle('DISP')
x= -100.00 y= 100.00 angulo= 180.00
octave:260> turtle('GIRA',90)
GIRA 90.000000
octave:261> turtle('AVANZA',100)
AVANZA 100.000000
octave:262> turtle('DISP')
x= -100.00 y=  0.00 angulo= 270.00
octave:263> turtle('GIRA',90)
GIRA 90.000000
octave:264> turtle('AVANZA',100)
AVANZA 100.000000
octave:265> turtle('DISP')
x=  0.00 y=  0.00 angulo=  0.00
octave:266>
```

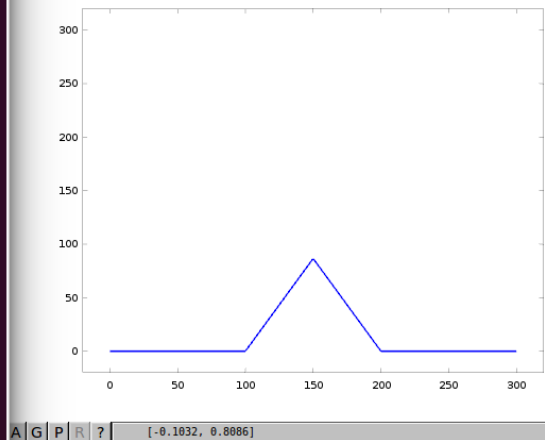
Una vez comprobado que la matemática de la tortuga funciona correctamente, podemos implementar las sentencias necesarias para que la tortuga dibuje los gráficos. En esta versión básica de tortuga la única función que realmente dibujará en pantalla será la función 'AVANZA'. Daremos por hecho que el programa principal que utilice nuestra función *turtle()* será el encargado de abrir y configurar la ventana de gráficos, haciendo un *hold on* para que la ventana de gráficos reciba los sucesivos comandos *plot()* enviados por la tortuga. Nuestra función 'turtle()' lo único que hace es enviar el comando *plot()* necesario para dibujar el segmento comprendido entre la posición de la tortuga al inicio del comando y la posición final calculada en base al ángulo y magnitud del avance de la tortuga. La función *avanza()* podría quedar resuelta de la siguiente forma:

```
function avanza(value)
    global x y angulo;
    fprintf('AVANZA %f\n', value)
    oldx = x;
    oldy = y;
    incx = value * cosd(angulo);
    incy = value * sind(angulo);
    x = oldx + incx;
    y = oldy + incy;
    xx = [oldx, x];
    yy = [oldy, y];
    plot(xx, yy, 'linewidth',2)
end
```

Podemos probar el resultado desde la consola de *Octave*. La secuencia de comandos y el gráfico que se obtendría

serían los siguientes:

```
octave:319> hold on
octave:320> xlim([-20,320])
octave:321> ylim([-20,320])
octave:322> turtle('INIT')
x= 0.00 y= 0.00 angulo= 0.00
octave:323> turtle('AVANZA', 100)
AVANZA 100.000000
octave:324> turtle('GIRA', 60); turtle('AVANZA',100)
GIRA 60.000000
AVANZA 100.000000
octave:325> turtle('GIRA', -120); turtle('AVANZA',100)
GIRA -120.000000
AVANZA 100.000000
octave:326> turtle('GIRA', 60); turtle('AVANZA',100)
GIRA 60.000000
AVANZA 100.000000
octave:327> █
```



Con esto tenemos creada la implementación básica de tortuga que nos permitirá realizar las primeras pruebas de dibujo. La forma habitual de utilizar la función `turtle()` será desde otras funciones que la utilizarán para realizar sus figuras. En el listado definitivo de la función `turtle()` que se muestra a continuación, se han suprimido las sentencias `fprintf()` de las funciones `avanza()` y `gira()`, para que no interfieran durante el dibujado de segmentos sucesivos.

```
function turtle(command, value)
% Tortuga básica para gráficos
% Necesita la existencia de unas variables globales x, y, angulo

global x y angulo;

if strcmp(command, 'GIRA')
    gira(value);
elseif strcmp(command, 'AVANZA')
    avanza(value);
elseif strcmp(command, 'DISP')
    disp();
elseif strcmp(command, 'INIT')
    init();
else
    dispError(command);
end
end

function gira(value)
global angulo;
% fprintf('GIRA %f\n', value)
angulo = angulo + value;
if abs(angulo) > 360
    angulo = rem(angulo, 360);
end
if angulo == 360 | angulo == -360
    angulo = 0;
end
if angulo < 0
    angulo = 360 + angulo;
end
end

function avanza(value)
```

```
global x y angulo;
% fprintf('AVANZA %f\n', value)
oldx = x;
oldy = y;
incx = value * cosd(angulo);
incy = value * sind(angulo);
x = oldx + incx;
y = oldy + incy;
xx = [oldx, x];
yy = [oldy, y];
plot(xx, yy, 'linewidth',2)
end

function disp()
global x y angulo;
fprintf('x= %6.2f y= %6.2f angulo= %6.2f\n', x, y, angulo)
end

function init()
global x y angulo;
x=0.0; y=0.0; angulo=0.0;
disp();
end

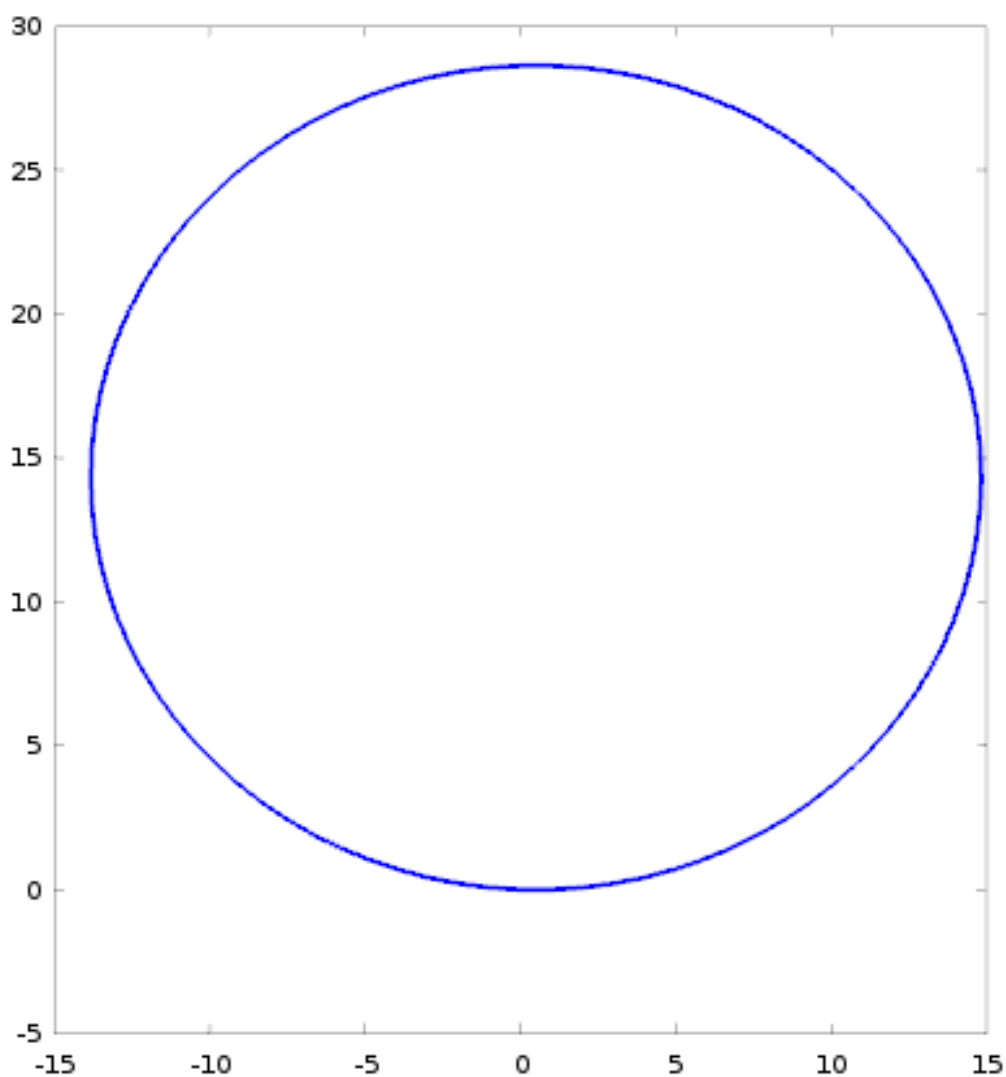
function dispError(command)
fprintf('Comando no reconocido: %s\n', command)
end
```

Podemos utilizar nuestra tortuga para dibujar algunas figuras sencillas. El siguiente listado muestra cómo utilizar la tortuga para dibujar un círculo, por ejemplo:

```
function circle()
% Dibuja un círculo utilizando la función turtle()
global x y angulo;
turtle('INIT');

close();
hold on;

for i = 0 : 90
    turtle('AVANZA', 1);
    turtle('GIRA',4);
end
end
```



En el siguiente ejemplo se utiliza la tortuga para dibujar una serie de triángulos:

```
function triang()
    % Dibuja triángulos
    global x y angulo;
    turtle('INIT');

    close();
    hold on;

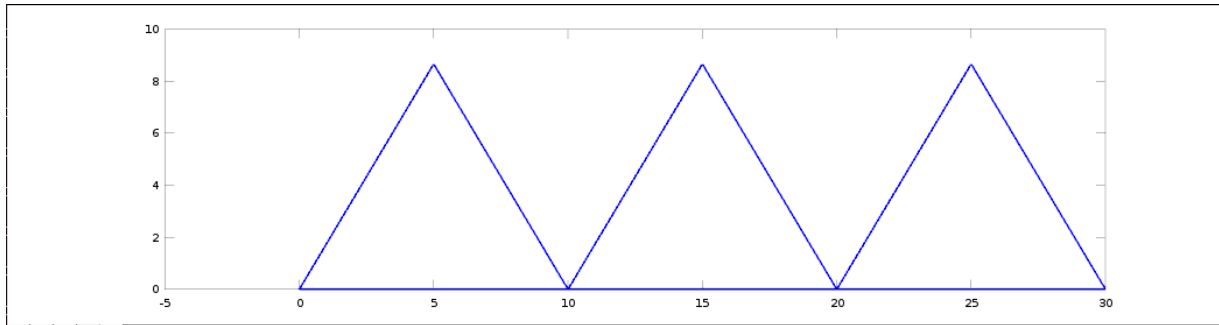
    for i = 1 : 3
        turtle('AVANZA', 10);
        turtle('GIRA', 120);
        turtle('AVANZA', 10);
```



```

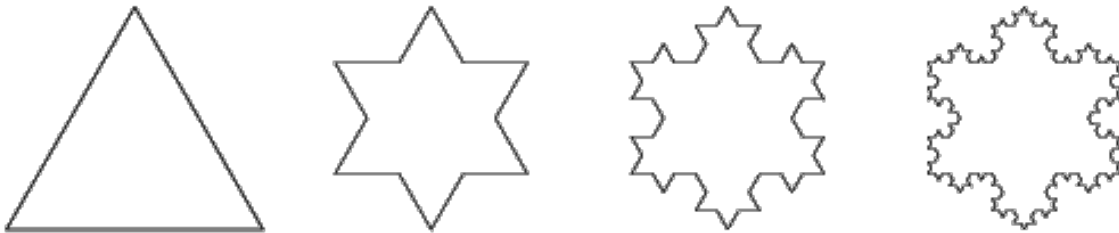
turtle('GIRA', 120);
turtle('AVANZA', 10);
turtle('GIRA', 120);
turtle('AVANZA', 10);
end
end

```



15.1 El copo de nieve de Koch

La curva fractal '*Koch snowflake*', también conocida como la '*isla de Koch*', fue descrita por primera vez por Helge von Koch en 1904. Se construye partiendo de un triángulo equilátero. En cada lado del triángulo se sustituye la tercera parte central por un nuevo triángulo equilátero de lado un tercio del primero. Este proceso se repite indefinidamente.



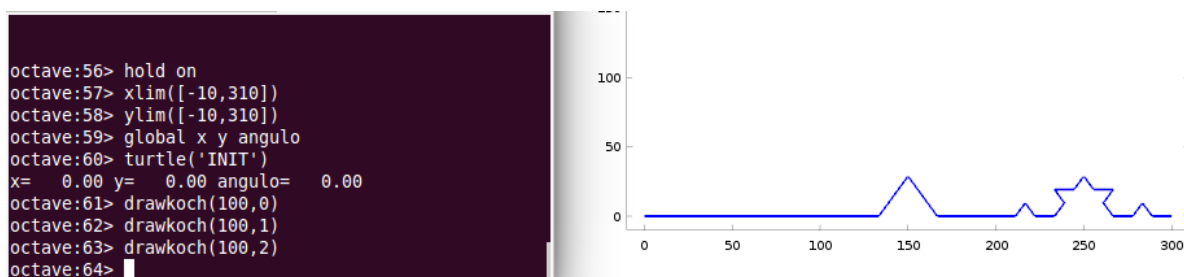
El resultado es una curva cuya longitud tiende a infinito, pero encerrando un área que tiende a una cantidad finita. Podemos implementar una función que dibuje una curva de Koch mediante la utilización de la tortuga creada en el apartado anterior. Para ello crearemos en primer lugar la función *drawkoch()* que dibuja, de forma recursiva, el segmento básico de la curva de Koch para un determinado nivel de profundidad:

```

function drawkoch(length, depth)
if depth == 0
    turtle('AVANZA', length);
else
    drawkoch(length/3, depth-1);
    turtle('GIRA', 60);
    drawkoch(length/3, depth-1);
    turtle('GIRA', -120);
    drawkoch(length/3, depth-1);
    turtle('GIRA', 60);
    drawkoch(length/3, depth-1);
end
end

```

Podemos probar la función recién creada enlazando los tres primeros niveles de segmentos de Koch. La secuencia de instrucciones y el resultado se puede apreciar en la siguiente figura:



Para dibujar el copo de nieve completo, con un determinado nivel de profundidad tendremos que dibujar tres tramos de Koch, cada uno girado -120 grados respecto del anterior. Podemos implementar una función que denominaremos `koch()`, que utiliza la función anterior para dibujar el copo de nieve. El código de la función `koch()` será el siguiente:

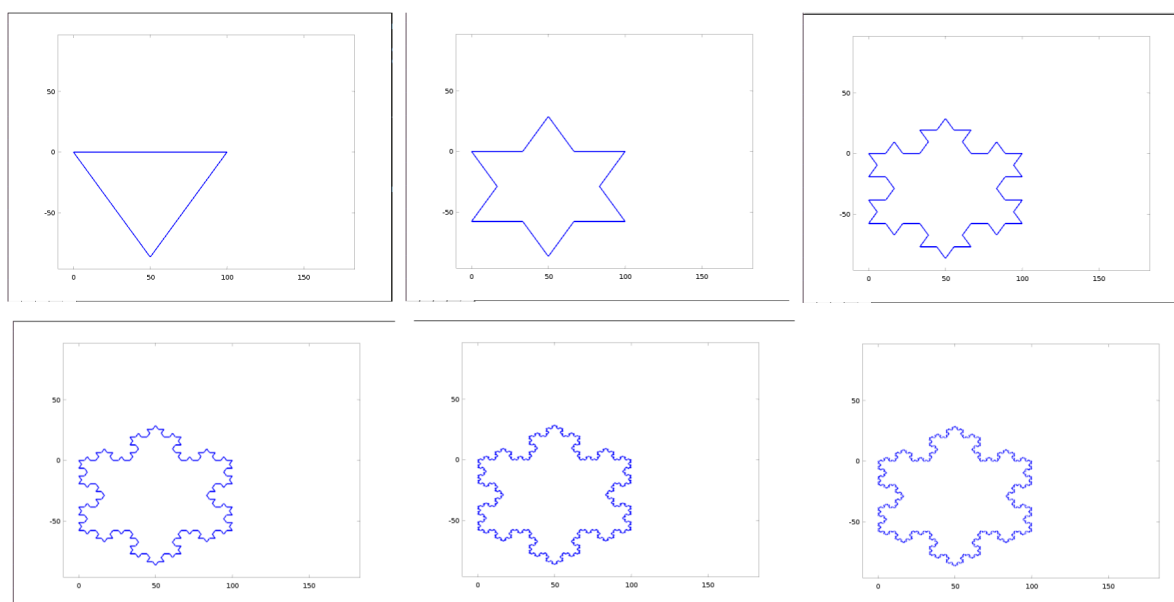
```
function koch(length, level)
% Dibuja la curva de Koch para un nivel determinado

    hold off;
    close;
    hold on;
    xlim([-10, 2*length*sind(60)+10]);
    ylim([-10-length*sind(60), 10+length*sind(60)]);

    global x y angle;
    turtle('INIT');

    drawkoch(length, level);
    turtle('GIRA', -120);
    drawkoch(length, level);
    turtle('GIRA', -120);
    drawkoch(length, level);
end
```

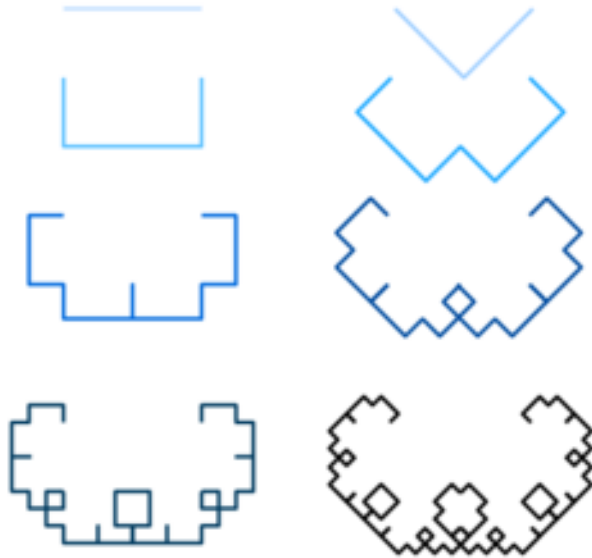
El resultado de la función para los primeros niveles se puede ver en la siguiente figura:



15.2 La curva C de Lévy

El último ejemplo que vamos a mostrar es el de la curva C de Lévy, descrita por primera vez por Ernesto Cesàro en 1906 y Georg Faber en 1910, pero que lleva el nombre del matemático francés Paul Lévy, que fue el primero que proporcionó un método geométrico para construirla.

La curva *C* comienza con un segmento recto de cierta longitud. Utilizando este segmento como hipotenusa, se construye un triángulo isósceles con ángulos 45° , 90° y 45° . Sustituimos el segmento original por los dos catetos del triángulo. El proceso se repite indefinidamente con cada uno de los segmentos que se generan. En la figura siguiente se muestran los primeros niveles de la curva *C*:



El código que se muestra a continuación dibuja la curva *C* para un determinado nivel de profundidad:

```
function ccurve(length,level)
    if level == 0
        turtle('AVANZA',length);
    else
        turtle('GIRA',45);
        ccurve(length/sqrt(2), level-1);
        turtle('GIRA',-90);
        ccurve(length/sqrt(2), level-1);
        turtle('GIRA',45);
    end
end
```

El resultado para el nivel de profundidad 12 se muestra en la siguiente figura:

```
octave:112> hold on
octave:113> global x y angulo
octave:114> turtle('INIT')
x=  0.00 y=  0.00 angulo=  0.00
octave:115> ccurve(100,12)
octave:116> █
```

