Procedimientos y figuras recursivas

Juan Álvarez

Las funciones devuelven un resultado que es utilizado en el lugar de invocación. Pero también hay funciones que ejecutan instrucciones pero que no devuelven un resultado. Estas funciones son llamadas procedimientos porque realizan acciones pero su objetivo no es devolver un valor. Por ejemplo, un procedimiento que recibe una fecha como un número entero de la forma DDMMAAAA y la escribe en la forma habitual se puede escribir de la siguiente manera:

```
#escribirFecha: int ->
#escribir fecha DDMMAAAA en la forma dia / mes / año
#ej: escribirFecha(30032023) escribe 30 / 3 / 2023
def escribirFecha(x):
  assert esFecha(x)
  dia=x//1000000; mes=x//10000%100; año=x%10000
  print(dia,'/',mes,'/',año)
  return
```

El contrato especifica que la función recibe un número entero y que no devuelve ningún resultado. De hecho, la instrucción return retorna a la instrucción siguiente a la invocación y se puede omitir si es la última de un procedimiento. Y dado que el procedimiento no devuelve ningún valor, no se puede probar con una instrucción assert y se debe invocar en la forma escribirFecha(n°).

Procedimientos recursivos que escriben en la pantalla

Un procedimiento puede invocarse recursivamente. Por ejemplo, una cuenta regresiva se puede programar de la siguiente manera:

```
#cuentaRegresiva: int ->
#escribir cuenta regresiva desde n >= 0
#ej: cuentaRegresiva(5) escribe 5, 4, 3, 2, 1
def cuentaRegresiva(n):
    assert type(n)==int and n>=0
    if n > 0:
        print(n)
        cuentaRegresiva(n-1)
    return
```

El contrato indica que la función recibe un número entero pero no entrega ningún resultado. Si el parámetro es positivo, la función lo escribe y se llama recursivamente con la instrucción cuentaRegresiva(n-1). La instrucción return al final del procedimiento (que se puede omitir) transfiere el control a la instrucción siguiente de la invocación recursiva anterior.

El procedimiento se puede reescribir en una forma que deja en evidencia que cuando el parámetro alcanza el valor cero, entonces la función retorna sin llamarse recursivamente:

```
def cuentaRegresiva(n):
    assert type(n)==int and n>=0
    if n==0: return
    print(n)
    cuentaRegresiva(n-1)
```

Una función puede también leer valores. Por ejemplo, una función que lee una lista de números y calcula sus factoriales puede establecer el siguiente diálogo que indica que el programa termina al leer un número negativo (puesto que la función factorial no está definida para números negativos):

```
n° entero?10
factorial: 3628800
…
n° entero?20
factorial: 2432902008176640000
n° entero?-1
```

El procedimiento, que no recibe parámetros, puede escribirse de la siguiente manera:

```
from factorial import *
#factoriales: ->
#leer numeros enteros (hasta un negativo) y escribir factoriales
#ej: factoriales()
def factoriales():
    n=int(input("n° entero?"))
    if n<0: return
    print("factorial:", factorial(n))
    factoriales()
```

La llamada recursiva se realiza con factoriales( ) sin pasar ningún argumento ni recibir ningún resultado. El caso base es la lectura de un número negativo y la instrucción return desencadena el retorno a las instrucciones return implícitas después de las llamadas recursivas previas.

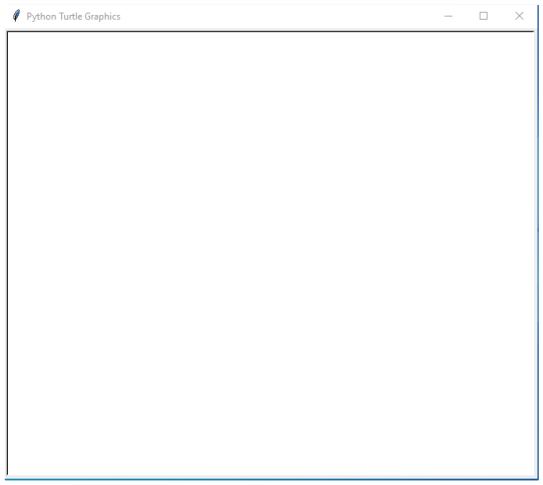
Figuras recursivas

Los procedimientos recursivos permiten graficar de manera muy sencilla figuras que tienen una forma con un patrón que se repite. En este caso se incluyen los polígonos regulares: triángulos equiláteros, cuadrados, pentágonos, hexágonos, etc. Por otra parte, están los fractales que son figuras geométricas cuya estructura básica se repite a diferentes escalas. De hecho, algunas estructuras naturales se pueden representar por fractales, por ejemplo, los copos de nieve.

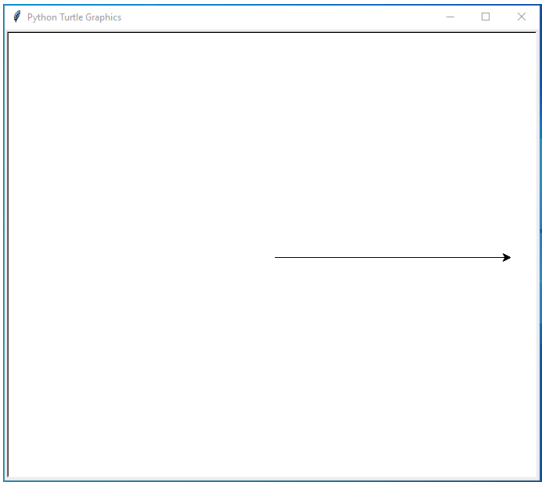
Módulo para graficar

El lenguaje Python tiene incorporado un módulo con procedimientos que permiten programar fácilmente los algoritmos recursivos para dibujar los polígonos regulares y los fractales. El módulo se llama turtle porque evoca a la tortuga del lenguaje Logo que en las décadas de los setenta y ochenta ayudaba a los alumnos de enseñanza básica a aprender geometría. En efecto, los niños y niñas aprendían geometría ingresando comandos para que una tortuga avanzara y girara en un plano.

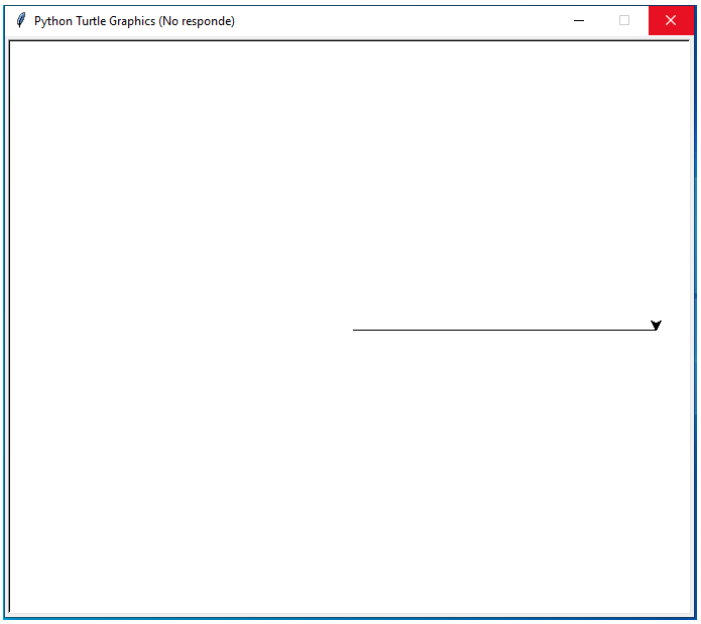
La instrucción from turtle import * deja disponibles los procedimientos del módulo predefinido turtle. El primero que se debe invocar es resetscreen() que abre una ventana en la pantalla y deja a la tortuga inicialmente ubicada en el centro de la ventana y "mirando a la derecha". La tortuga aún no es visible, pero está apuntando en dirección horizontal hacia la derecha de la ventana (desde el punto de vista de quien observa).



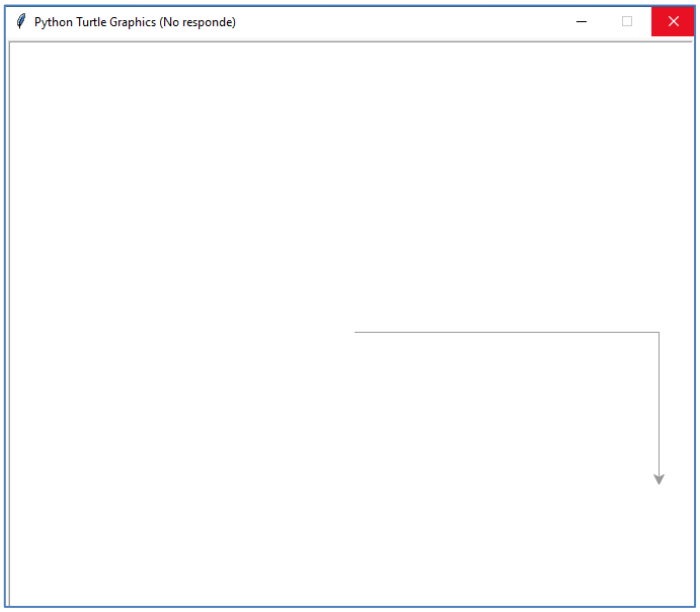
Con la invocación forward(300) la tortuga avanza en 300 pixeles en la dirección en que apuntaba la tortuga, en este caso a la derecha. El avance queda graficado con una línea en la ventana y la nueva posición y dirección de la tortuga queda indicada con una punta de flecha:



Con la invocación right(90) la tortuga gira en 90 grados a la derecha por lo tanto queda apuntando hacia abajo de la ventana. La nueva dirección u orientación de la tortuga queda reflejada en una punta de flecha. Existe la función left para girar la tortuga hacia la izquierda y cuyo argumento debe también expresarse en grados.

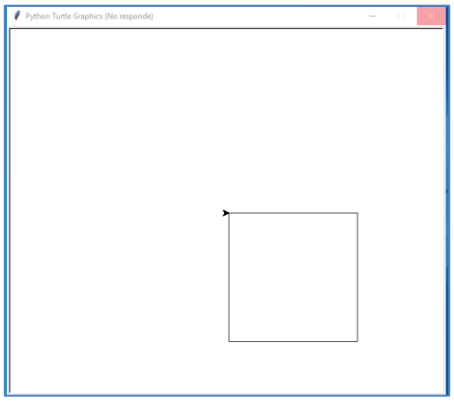


En esta nueva ubicación de la tortuga se puede dar un nuevo comando, por ejemplo forward(150) para que la tortuga avance 150 pixeles en la dirección en que está apuntando. De esta manera, con las funciones ressetscreen, forward, right y left se pueden dibujar diferentes figuras en la ventana.



Polígonos regulares

Utilizando las funciones del módulo turtle se puede dibujar un cuadrado haciendo 4 avances y 4 giros. Por ejemplo, un cuadrado de 200 pixeles se mostraría en la ventana en la forma:



Con el propósito de dibujar cuadrados de distintos tamaños es conveniente escribir un procedimiento que reciba el tamaño del lado como parámetro.

```
from turtle import *
resetscreen()
#cuadrado: int ->
#dibujar cuadrado de lado L pixeles
#ej: cuadrado(200)
def cuadrado(L):
    assert type(L)==int and L>=2
    forward(L); right(90)
    forward(L); right(90)
    forward(L); right(90)
    forward(L); right(90)
```

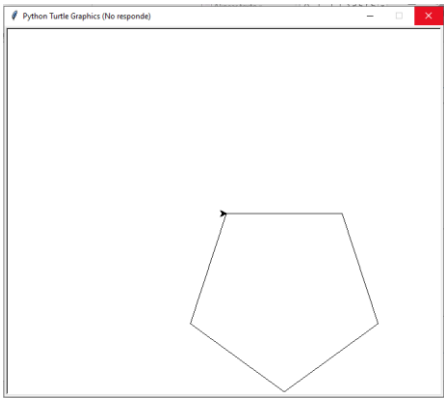
Para no repetir 4 veces un avance y un giro, el procedimiento que dibuja un cuadrado puede implementarse invocando un procedimiento interno que dibuja recursivamente los 4 lados:

```
def cuadrado(L):
    assert type(L)==int and L>=2
    #dibuja n lados de largo L
    def lados(n,L):
        if n==0: return
        forward(L); right(90)
        lados(n-1,L)
    lados(4,L)
```

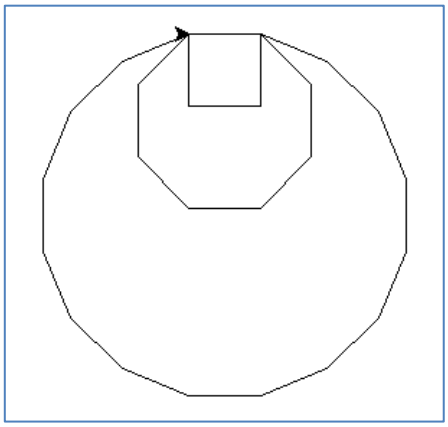
El procedimiento que dibuja un cuadrado con una función interna recursiva puede generalizarse de modo que dibuje un polígono regular de N lados haciendo giros de 360/N grados:

```
from turtle import *
resetscreen()
#poligono: int int ->
#dibujar poligono regular de N lados de L pixeles
#ej: poligono(5,180)
def poligono(N,L):
    assert type(N)==int and N>=3
    assert type(L)==int and L>=2
    #dibuja n lados largo L girando en angulo
    def lados(n,L,angulo):
        if n==0: return
        forward(L); right(angulo)
        lados(n-1,L,angulo)
    lados(N,L,360/N)
```

Por ejemplo, para dibujar un pentágono, es decir un polígono regular de 5 lados de tamaño 180 pixeles, se invoca poligono(5,180) y en la ventana aparecerá la siguiente figura:

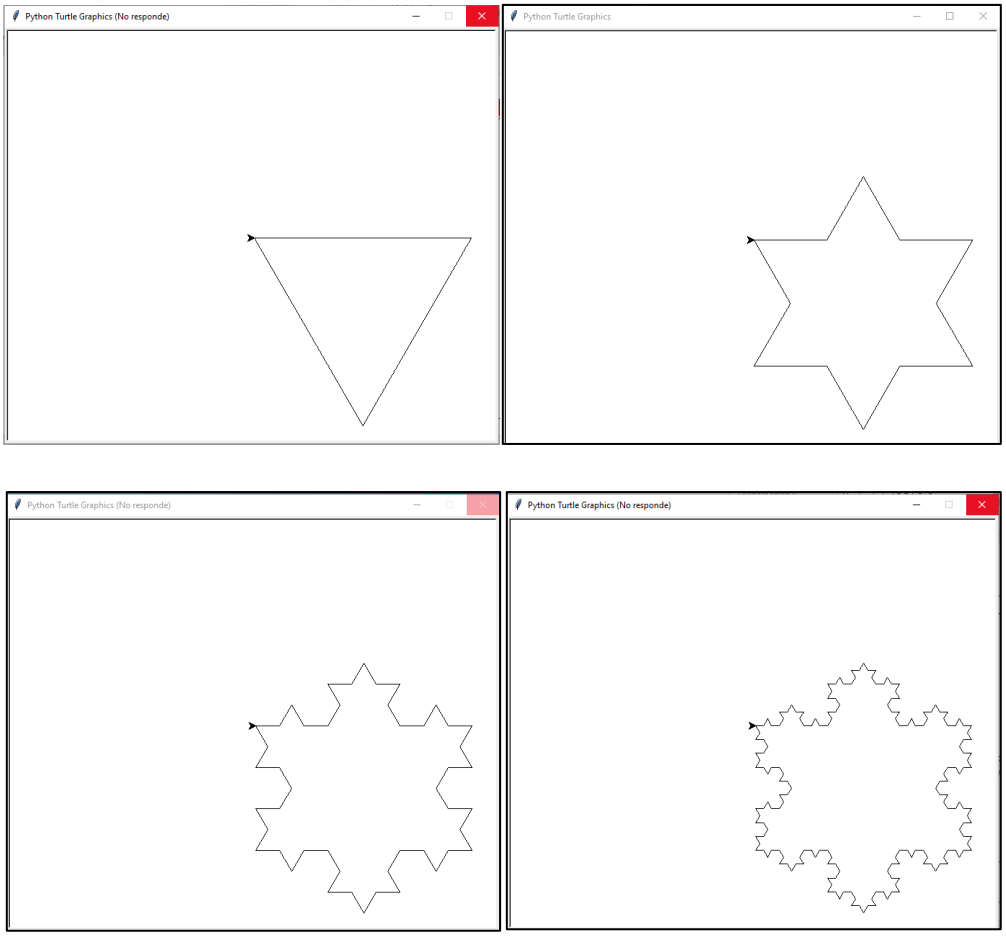


Al invocar el procedimiento polígono sucesivamente aparecen figuras superpuestas. Por ejemplo poligono(4,50), poligono(8,50) y poligono(16,50) muestra:



Fractales

Los fractales son figuras geométricas cuya estructura básica se repite a diferentes escalas. Por ejemplo, el patrón del fractal de Koch se aprecia al observar los fractales de orden 1, 2 3 y 4 que se producen con fractalKoch(1,300) hasta fractalKoch(4,300) que parece un “copo de de nieve”.



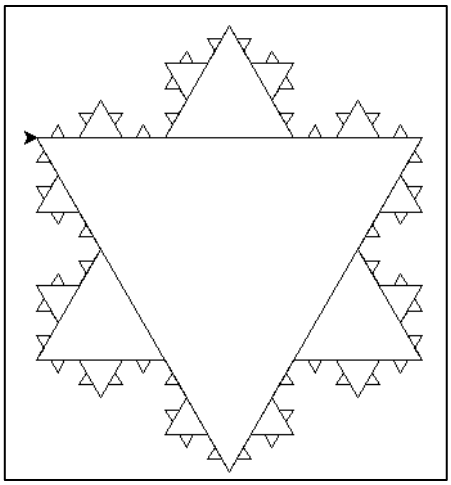
El fractal de Koch de orden 1 es un triángulo equilátero. En el fractal de orden 2, cada lado del triángulo se divide en 4 segmentos: _/_. En los fractales de orden superior ocurre lo mismo con los lados de los fractales de orden inmediatamente inferior. El procedimiento que implementa este patrón recursivamente se puede escribir de la siguiente manera:

```
from turtle import *
resetscreen()
#fractalKoch: int int ->
#dibujar fractal de Koch de orden n
#ej: fractalKoch(2,300)
def fractalKoch(n,L):
    assert type(n)==int and n>=1
    assert type(L)==int and L>=6
```

```
#dibuja lado de fractal de Koch de orden n
def lado(n,L):
    if n==1 or L<6:
        #dibuja 1 linea
        forward(L)
    else:
        #dibuja 4 lados: __/\__
        lado(n-1,L/3); left(60)
        lado(n-1,L/3); right(120)
        lado(n-1,L/3); left(60)
        lado(n-1,L/3)
#dibuja triangulo equilatero como base
lado(n,L); right(120)
lado(n,L); right(120)
lado(n,L); right(120)
```

El procedimiento fractalKoch "dibuja" un triángulo equilátero con 3 lados y 3 giros de 120 grados. El dibujo de cada lado se realiza invocando al procedimiento recursivo de nombre lado. El caso base (orden 1) dibuja efectivamente una línea con la función forward. Si el orden es superior a 1 se "dibujan" los 4 segmentos _/_ en que se divide un lado invocando recursivamente a la misma función. La condición L<6 del caso base garantiza que la función no se siga invocando recursivamente cuando ya no se pueda dividir un segmento en tres partes.

La repetición del patrón de dibujo del fractal de Koch, hasta alcanzar el "copo de nieve", se aprecia invocando sucesivamente al procedimiento: fractalKoch(1,300); fractalKoch(2,300); fractalKoch(3,300); fractalKoch(4,300). Después de dibujar cada fractal la tortuga queda en la misma posición original permitiendo superponer los dibujos:



En resumen, para dibujar un fractal se debe "dibujar" la figura base. Cada lado de la figura base se debe "dibujar" de acuerdo al patrón que se repite a través de un procedimiento recursivo que en su caso base dibuja efectivamente una línea.