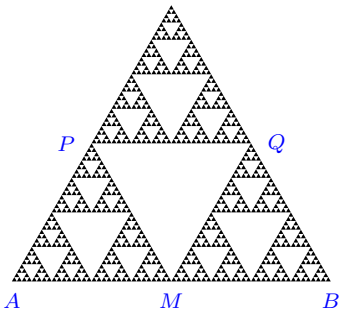
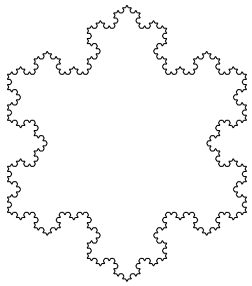
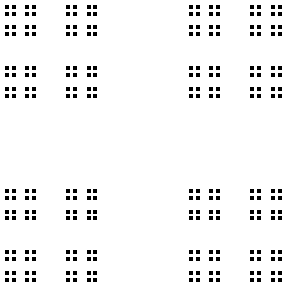
Práctica 5 – Fractales
20 % de la nota de prácticas
Fecha límite de entrega: 5 de mayo

Los fractales son objetos geométricos que se repiten *recursivamente* a diferentes escalas, es decir, son la unión de copias más pequeñas de sí mismos que cumplen a su vez esta propiedad. El término *fractal* (del latín *fractus*, quebrado) lo acuñó el matemático Benoît Mandelbrot en 1975, pero estos *monstruos* (en palabras de Poincaré) empezaron a estudiarse con cierta aprensión algún tiempo antes. Los siguientes son tres ejemplos de fractales bien conocidos.



Triángulo de Sierpiński

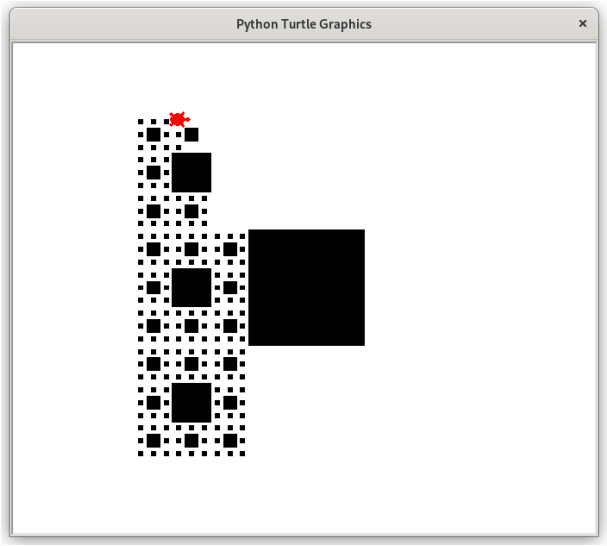

Copo de nieve de Koch


Polvo de Cantor

Centrémonos en el triángulo de Sierpiński y observemos que está compuesto por tres copias de sí mismo a mitad de tamaño, dos de ellas sobre su misma base y la tercera apoyada sobre estas dos. Concretamente, el triángulo de Sierpiński con base en el segmento AB consiste en tres triángulos de Sierpiński con bases en AM, MB y PQ. Esta propiedad la cumplen a su vez cada uno de estos triángulos pequeños, ajustados convenientemente los segmentos según la relación de semejanza (es decir, según la translación, rotación y escala aplicadas). Sin embargo, si hacemos zoom lo suficiente veremos que las copias del fractal a partir de cierto tamaño se han pintado como simples triángulos rellenos. Obviamente, si queremos pintar un fractal en la práctica tenemos que conformarnos con una aproximación y para ello es razonable terminar la recurrencia tras cierto número de pasos y pintar una forma geométrica simple en su lugar.


$n = 0$ $n = 1$ $n = 2$ $n = 3$ $n = 4$

En esta práctica vamos a construir aproximaciones de algunos fractales en el plano mediante funciones recursivas. Estas funciones reproducirán la definición recursiva del fractal hasta alcanzar el número de pasos escogido y entonces dibujarán un triángulo, un cuadrado o la forma geométrica simple que más convenga como aproximación, como se ve en la figura anterior.



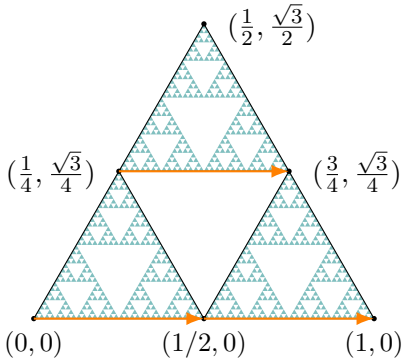
Usaremos para dibujar el paquete `turtle` de la biblioteca estándar de Python. Este paquete está inspirado en el lenguaje educativo Logo y permite controlar una *tortuga* sobre un lienzo mientras dibuja líneas, rellena figuras y añade otros elementos. La biblioteca es muy sencilla de utilizar, pues está pensada para niños. Sin embargo, no será necesario hacerlo directamente porque proporcionamos la siguiente función `dibuja` que dada una secuencia de puntos dibuja el segmento o polígono relleno que estos definen.

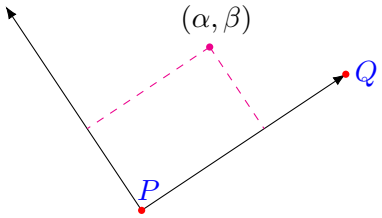
```
Punto = tuple[float, float]

def dibuja(puntos: Sequence[Punto]):
    """Dibuja un segmento o polígono dada una secuencia de puntos en el plano"""

    # Se coloca en la posición inicial
    turtle.up()  # levanta el lápiz
    turtle.goto(points[0][0], puntos[0][1])
    turtle.down()  # baja el lápiz

    # Rellena la figura determinada por los puntos
    turtle.begin_fill()
    for x, y in puntos[1:]:
        turtle.goto(x, y)
    turtle.end_fill()
```

El movimiento de la tortuga por el lienzo es una animación y es posible que se haga demasiado lento cuando hay mucho que pintar. La velocidad de la tortuga se puede ajustar con las funciones `turtle.speed('fastest')` y especialmente con `turtle.tracer`.

1 Clase Segmento para describir el fractal

La relación de recurrencia de un fractal es un conjunto de semejanzas que lo transforman en las copias de sí mismo de las que está formado. Como hemos visto con el ejemplo, un segmento orientado sirve para caracterizar la posición y el tamaño de cualquier copia de un fractal en el plano, indicando la base sobre la que se construye. Para describir la recurrencia basta entonces especificar cómo obtener las bases de las copias reducidas a partir del segmento sobre el que se apoya el fractal completo.





En el caso del triángulo de Sierpiński sobre el segmento de $(0,0)$ a $(1,0)$, es sencillo encontrar los extremos de los segmentos sobre los que construir las copias más pequeñas. Sin embargo, la fórmula se volvería más complicada si quisiéramos construirlo sobre un segmento arbitrario (p_1, p_2) a (q_1, q_2), como los que saldrán al expandir recursivamente el fractal. Afortunadamente, considerando un sistema de referencia local, podemos razonar como si nuestro fractal siempre estuviera apoyado en el segmento $(0,0)$ a $(1,0)$. Para que el cambio de coordenadas nos distraiga lo menos posible, vamos a programar una clase `Segmento` con dos métodos `punto_relativo` y (segmento) `relativo` que harán la traducción del sistema de referencia local del segmento al global del plano.

```
class Segmento:
    """Segmento orientado de recta"""

    def __init__(self, inicio: Punto, fin: Punto):
        """Segmento entre dos puntos dados en coordenadas globales"""
        self.inicio = inicio
        self.vector = (fin[0] - inicio[0], fin[1] - inicio[1])
```

```
    def punto_relativo(self, punto_local: Punto) -> Punto:
        """Coordenadas globales de punto_local, que está expresado en
           el sistema de coordenadas del segmento actual (self)"""
        pass  # aquí va tu código

    def relativo(self, inicio: Punto, fin: Punto) -> 'Segmento':
        """Segmento entre dos puntos dados en coordenadas locales"""
        return Segmento(self.punto_relativo(inicio),
                        self.punto_relativo(fin))
```

Tu tarea en esta clase consiste únicamente en implementar `punto_relativo`. El atributo `inicio` almacena el punto de inicio del segmento y `vector` se corresponde con el vector que lo une con el otro extremo. Si `inicio` es (x, y) y `vector` es (u, v), podemos tomar como base ortogonal $\{(u, v), (-v, u)\}$ y transformar un punto en coordenadas locales (α, β) en $(x + \alpha u - \beta v, y + \alpha v + \beta u)$, como se ve en la ilustración anterior.

Ahora, para cualquier segmento PQ, el objeto `Segmento(`P`, `Q`)` permite referirnos a su punto medio como $(1/2, 0)$, al vértice superior del triángulo equilatero sobre PQ como $(1/2, \sqrt{3}/2)$, etcétera, sin preocuparnos de las coordenadas absolutas de P y Q.

2 Aproximación de algunos fractales

Ahora vamos a programar funciones recursivas para dibujar aproximaciones de tres ejemplos de fractales, utilizando la clase `Segmento` que acabamos de introducir y la función `dibuja`. La definición del triángulo de Sierpiński se proporciona como un ejemplo, mientras que las otras dos definiciones las has de programar tú. Todas tendrán la forma

```
def nombre_fractal(base: Segmento, n: int)
```

donde `base` es la base sobre la que se dibujará el fractal y `n` es el número de pasos de la aproximación. Es necesario que la implementación sea recursiva, con un caso base $n = 0$ en el que se pintará el segmento o polígono que mejor aproxime la forma del fractal. En el caso recursivo se habrán de calcular los segmentos sobre los que se apoyan las siguientes copias autosemejantes para llamar a la función recursivamente sobre ellos.

2.1 Triángulo de Sierpiński

El triángulo de Sierpiński recibe el nombre del matemático polaco Wacław Sierpiński (1882-1969), que lo describió en 1915, aunque ya se utilizaba como elemento decorativo desde al menos el siglo XIII en la Italia medieval. Resumiendo las explicaciones anteriores y usando coordenadas locales respecto al segmento sobre el que se construye, dibujar el fractal consiste en:

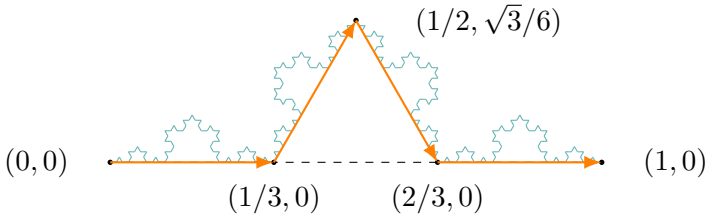
- Si $n = 0$, aproximarlo por un triángulo equilatero, de vértices los del segmento base y $(1/2, \sqrt{3}/2)$.
- Si $n \geq 1$, descomponerlo en tres fractales apoyados sobre los segmentos $(0,0)(1/2,0)$, $(1/2,0)(1,0)$ y $(1/4,\sqrt{3}/4)(3/4,\sqrt{3}/4)$.

Se incluye el código de la función recursiva como ejemplo.

```python
def triángulo_sierpinski(base: Segmento, n: int):
    """Dibuja el triángulo de Sierpiński"""

    A, B, h = (0, 0), (1, 0), (3 ** 0.5) / 2  # altura del triángulo

    if n == 0:  # caso base, pintamos un triángulo relleno
        vértices = (A, B, (1/2, h))
        dibuja([base.punto_relativo(v) for v in vértices])

    else:  # caso inductivo
        M, P, Q = (1/2, 0), (1/4, h/2), (3/4, h/2)
        triángulo_sierpinksi(base.relativo(A, M), n - 1)
        triángulo_sierpinksi(base.relativo(M, B), n - 1)
        triángulo_sierpinksi(base.relativo(P, Q), n - 1)
```

2.2 Curva de Koch

El copo de nieve de Koch que aparece en la primera página del enunciado es la composición de tres *curvas de Koch* construidas sobre los lados de un triángulo equilatero. La curva de Koch fue descrita en 1904 por el matemático sueco Helge von Koch (1870-1924).

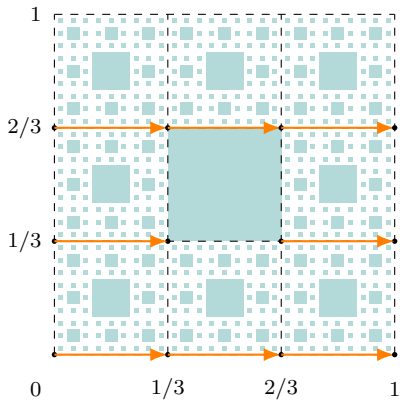


Esta curva fractal es la unión de otras cuatro copias de sí misma apoyadas sobre sendos segmentos que se muestran en naranja en el diagrama anterior. La aproximación simple más razonable de la curva es el segmento $(0,0)(1,0)$.

Programa las funciones `curva_koch` y `copo_koch` para dibujar las aproximaciones de la curva y el copo de nieve de Koch sobre el segmento y con el número de pasos dados. El vértice inferior del copo debe coincidir con el punto medio del segmento base y este ha de ser paralelo al lado opuesto del triángulo (véase la primera página). La función `copo_koch` no debe ser recursiva, sino que ha de delegar en `curva_koch`.

2.3 Alfombra de Sierpiński

La alfombra de Sierpiński (1916), al igual que el polvo de Cantor que aparece en la primera página del enunciado, es una generalización en el plano del conjunto de Cantor.



Una alfombra de Sierpiński está compuesta recursivamente por 8 alfombras de Sierpiński colocadas sobre todos los cuadrantes de una rejilla 3 por 3 salvo el central. Por motivos estéticos y para dar un poco de variedad a la práctica, esta vez no vamos a pintar el fractal sino su complementario respecto al cuadrado con base en el segmento dado. De esta forma, en cada paso se pintará el rectángulo central que se excluye del fractal, y no se pintará nada cuando $n = 0$. Escribe una función recursiva `alfombra_sierpinski`

```
def alfombra_sierpinski(base: Segmento, n: int) -> float
```

con los mismos argumentos que las anteriores que haga esto y además devuelva el área total pintada, que también se ha de calcular recursivamente.

Recapitulación

Tu tarea consiste en implementar:

1. el método `punto_relativo` de la clase `Segmento`,
2. las funciones `curva_koch` y `copo_koch` para dibujar aproximaciones de la curva y el copo de nieve de Koch, respectivamente, y
3. la función `alfombra_sierpinski` para pintar aproximaciones del complementario de la alfombra de Sierpiński y calcular sus áreas.