9.1 O *maior divisor comum* de dois números inteiros positivos pode ser calculado recursivamente pelo *algoritmo de Euclides*:

$$\begin{array}{rcll} mdc(a,\, b) & = & a & \quad (\text{se } b = 0) \\ mdc(a,\, b) & = & mdc(b, a \bmod b) & \quad (\text{se } b \neq 0) \end{array}$$

Implemente este algoritmo como uma função recursiva `mdc(a,b)` em Python. Recorde que o resto da divisão inteira $a \bmod b$ se escreve como `a%b`.

▷ **9.2** Dois inteiros são *co-primos* se e só se o seu maior divisor comum for 1. Por exemplo: 4 e 15 são co-primos, mas não o são 4 e 14 (porque 2 é um divisor comum). Defina uma função `coprimos(n)` cujo resultado é a lista de inteiros entre 1 e n que são co-primos com n. Exemplos:

```
>>> coprimos(6)
[1, 5]
>>> coprimos(7)
[1, 2, 3, 4, 5, 6]
>>> coprimos(14)
[1, 3, 5, 9, 11, 13]
```

Sugestão: use a função `mdc(a,b)` do exercício anterior como auxiliar; note que a função `coprimos(n)` não têm de ser recursiva.

9.3 Recorde a seguinte fórmula de recorrência para calcular o *coeficiente binomial* $\binom{n}{k}$ apresentada num exercícios da folha 6:

$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad (\text{se } n > 0 \text{ e } 0 < k < n)$$

Traduza esta recorrência numa definição recursiva duma função `binom(n,k)` para calcular coeficientes binomiais.

9.4 Traduza para Python a seguinte definição recursiva da *função de Ackermann*:

$$ack(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ ack(m-1,1) & \textit{se } m > 0 \text{ e } n = 0 \\ ack(m-1, ack(m,n-1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

Atenção: esta função cresce muito rapidamente, pelo que só a conseguirá calcular no computador para valores de m, n pequenos. Por exemplo: $ack(3,2) = 29$ mas $ack(4,2)$ é um número com 19729 algarismos.

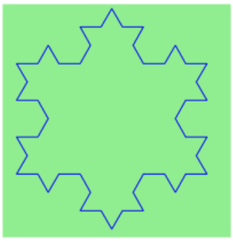
Pode conferir alguns valores tabelados na página da Wikipedia: `http://en.wikipedia.org/wiki/Ackermann_function`.

▷ **9.5** Recorde que um *palíndromo* é uma cadeia de texto que é igual quando lida da esquerda para a direita e vice-versa. Para testar se uma cadeia é um palíndromo podemos usar o seguinte algoritmo recursivo:

1. *se a cadeia é vazia ou tem um só carater* (comprimento 0 ou 1) então é sempre um palíndromo;
2. *se a cadeia tem 2 ou mais carateres* então será um palíndromo se e só se o primeiro e último carateres são iguais e ainda se a sub-cadeia sem esses dois carateres for também um palíndromo.

Implemente este algoritmo como uma função recursiva `palindromo(txt)`.

9.6 Utilizando a função `koch(n,side)` apresentada na aula téorica 15, defina uma função `floco(n,side)` para desenhar o floco de neve de Koch, como ilustrado na figura seguinte.



9.7 Considere a função recursiva `draw_tree()` para desenhar árvores apresentada na aula teórica 16.

(a) Use a função `pencolor()` do módulo turtle para alterar a cor de todos os ramos para castanho, exceto os últimos, que serão verdes.

(b) Use a função `pensize()` do módulo turtle para alterar a espessura dos ramos para que fique menor à medida que o seu comprimento diminui.

(c) Altere o ângulo de rotação de cada ramo para um valor aleatório entre 15 e 45 graus.

(d) Altere a dimensão dos ramos para que seja diminuída por um fator aleatório entre $\pm 10\%$ em cada bifuração.

9.8 Um *triângulo de Sierpinski* de ordem 0 é um triângulo equilátero. Um triângulo de ordem 1 é constituído por 3 triângulos mais pequenos (ligeiramente separados na figura em baixo para facilitar a compreensão). Ilustramos ainda triângulos de Sierpinski de ordem 2 e 3.

Defina uma função recursiva `sierpinski(n,lado)` para desenhar um triângulo de Sierpinski de ordem e lado dados.