Arbres_Pythagore

August 24, 2023

1 Arbres de Pythagore

Marc Lorenzi

10 août 2023

[1]:
```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
import math
import random
```

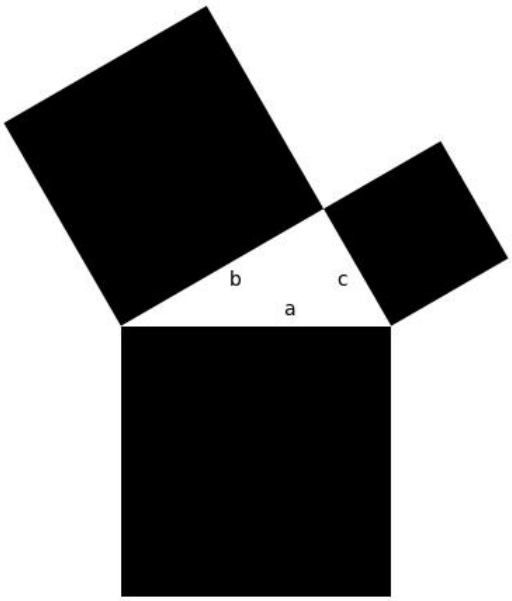
2 1. Introduction

2.0.1 1.1 Le théorème de Pythagore

Le théorème de Pythagore affirme que dans un triangle rectangle dont la longueur de l'hypoténuse est a et les longueurs des côtés de l'angle droit sont b et c, on a

$$a^2 = b^2 + c^2$$

On représente souvent géométriquement ce résultat par la figure ci-dessous.



Le grand carré est de côté a, les petits carrés de côtés b et c. L'aire du grand carré, a^2, et égale à la somme, $b^2 + c^2$, des aires des petits carrés.

Qu'arrive-t-il si on réapplique le théorème de Pythagore aux petits carrés ? C'est ce que nous allons voir.

2.0.2 1.2 Vecteurs

Pour simplifier le code des fonctions qui vont suivre, écrivons une classe `Vecteur`. Cette classe implémente le minimum : addition et soustraction de deux vecteurs, et produit d'un réel par un vecteur. Les coordonnées d'un vecteur u sont $u.x$ et $u.y$.

[2]:
```
class Vecteur:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self): return '(' + str(self.x) + ', ' + str(self.y) + ')'

    def __add__(self, v):
        return Vecteur(self.x + v.x, self.y + v.y)
```

```
    def __sub__(self, v):
        return Vecteur(self.x - v.x, self.y - v.y)

    def __rmul__(self, t):
        return Vecteur(t * self.x, t * self. y)
```

[3]:
```
w = Vecteur(2, 3) + Vecteur(3, 7)
print(w)
```

```
(5, 10)
```

La fonction `base` renvoie une base orthonormée directe (u, v) du plan. Le réel θ est l'angle orienté entre l'axe Ox et le vecteur u.

[4]:
```
def base(theta):
    c, s = math.cos(theta), math.sin(theta)
    u = Vecteur(c, s)
    v = Vecteur(-s, c)
    return (u, v)
```

[5]:
```
print(base(math.pi / 3))
```

```
((0.5000000000000001, 0.8660254037844386), (-0.8660254037844386,
0.5000000000000001))
```

2.0.3 1.3 Compteurs

Pour pouvoir compter le nombre d'opérations effectuées par certaines des fonctions que nous allons écrire, créons une classe `Compteur`. Un compteur possède une méthode `incr` pour l'incrémenter et une méthode `reset` pour le remettre à zéro.

[6]:
```
class Compteur():

    def __init__(self): self.val = 0
    def incr(self): self.val += 1
    def reset(self): self.val = 0
```

Créons un compteur global `cptr`.

[7]:
```
cptr = Compteur()
```

2.0.4 1.4 Carrés

Nous allons passer notre temps à afficher des carrés. Un carré $(ABCD)$ dans le plan est caractérisé par

- Son « premier sommet » A.
- La longueur ℓ de ses côtés.

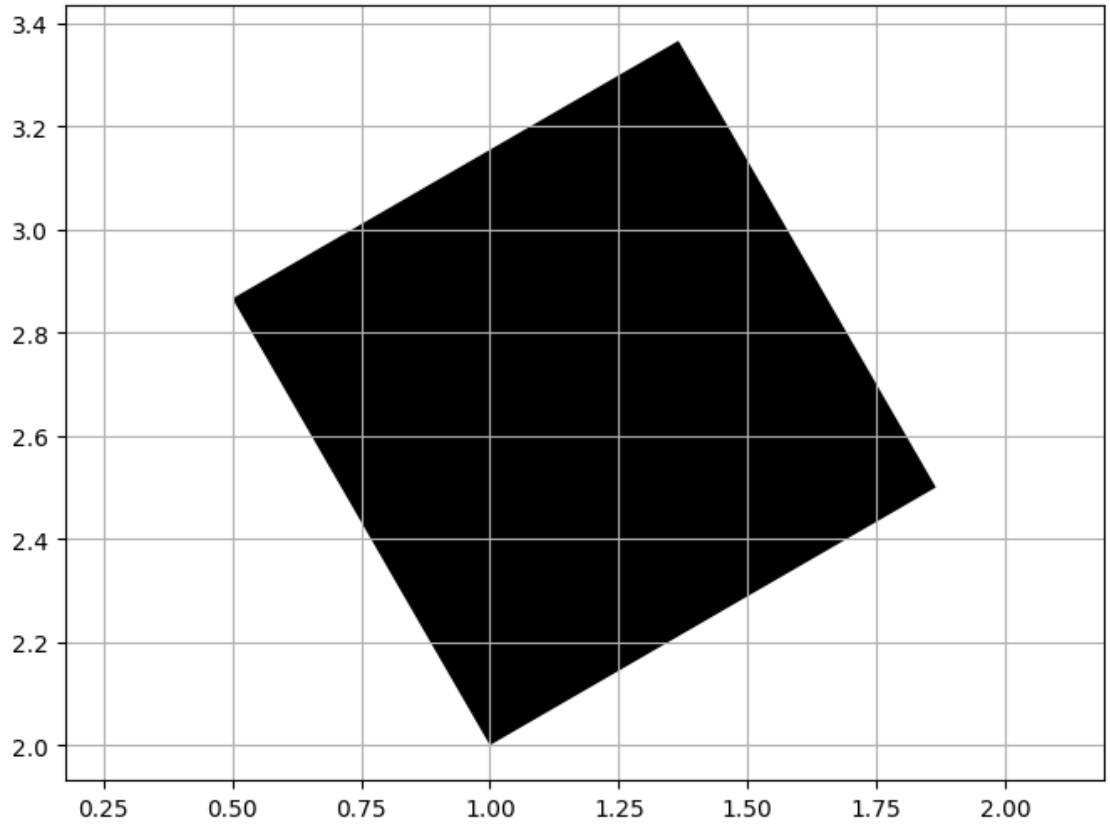
- L'angle θ entre l'axe Ox et son « premier côté » $\overrightarrow{AB}$.

La fonction `plot carre` prend en paramètre ces 3 quantités, ainsi qu'une couleur. Elle affiche le carré et incrémente le compteur `cptr`.

[8]:
```
plt.rcParams['figure.figsize'] = (8, 6)
```

[9]:
```
def plot_carre(A, l, theta, couleur):
    cptr.incr()
    u, v = base(theta)
    B = A + l * u
    C = B + l * v
    D = C - l * u
    P = Polygon([(A.x, A.y), (B.x, B.y), (C.x, C.y), (D.x, D.y)],␣
 ↪facecolor=couleur)
    plt.gca().add_patch(P)
    #plt.fill([A.x, B.x, C.x, D.x], [A.y, B.y, C.y, D.y], couleur)
```

[10]:
```
plot_carre(Vecteur(1, 2), 1, math.pi / 6, 'k')
plt.axis('equal')
plt.grid()
```

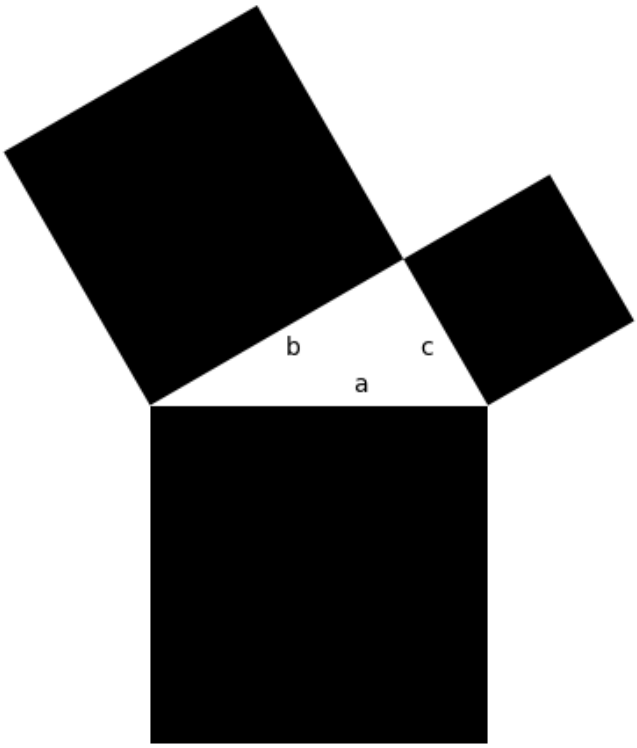


2.0.5 1.4 Le théorème de Pythagore

À titre d'exercice, reproduisons la figure du début du notebook. La fonction `test_pythagore` prend en paramètres les trois quantités caractérisant le grand carré, ainsi qu'un angle φ qui est l'angle entre le côté « haut » du grand carré et le côté « bas » du petit carré de gauche. On doit avoir $0 < \varphi < \frac{\pi}{2}$.

[11]:
```
def test_pythagore(A, l, theta, phi):
    plot_carre(A, l, theta, 'k')
    u, v = base(theta)
    l1 = l * math.cos(phi)
    A1 = A + l * v
    plot_carre(A1, l1, theta + phi, 'k')
    u, v = base(theta + phi)
    A2 = A1 + l1 * u
    l2 = l * math.sin(phi)
    plot_carre(A2, l2, theta + phi - math.pi / 2, 'k')
```

[12]:
```
test_pythagore(Vecteur(1, 1), 1, 0, math.pi / 6)
plt.text(1.6, 2.04, 'a')
plt.text(1.4, 2.15, 'b')
plt.text(1.8, 2.15, 'c')
plt.axis('equal')
plt.axis('off')
plt.savefig('theo_pythag.jpg')
#plt.grid()
```



2.1 2. Arbres de Pythagore

2.1.1 2.1 Un bon début

Reprenons l'illustration ci-dessus du théorème de Pythagore. Nous pouvons recommencer l'opération Pythagore sur les deux petits carrés obliques. Il suffit pour cela de reprendre le code de la fonction `test_pythagore` et de remplacer les deux appels à `plot_carre` par deux appels à `test_pythagore`.

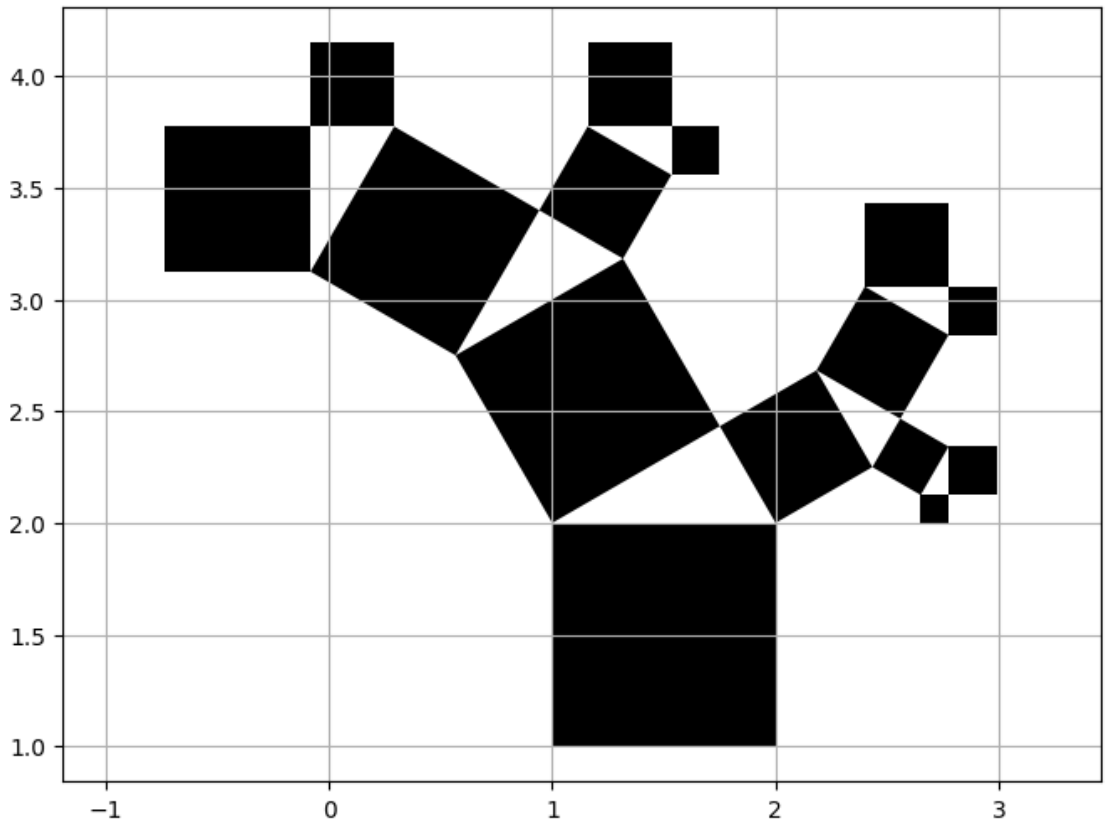
[13]:
```
def test_pythagore2(A, l, theta, phi):
    plot_carre(A, l, theta, 'k')
    u, v = base(theta)
    l1 = l * math.cos(phi)
    A1 = A + l * v
    test_pythagore(A1, l1, theta + phi, phi)
    u, v = base(theta + phi)
    A2 = A1 + l1 * u
    l2 = l * math.sin(phi)
    test_pythagore(A2, l2, theta + phi - math.pi / 2, phi)
```

[14]:
```
test_pythagore2(Vecteur(1, 1), 1, 0, math.pi / 6)
plt.axis('equal')
plt.grid()
```

Évidemment, nous pouvons maintenant écrire une fonction `test_pythagore3` qui effectue deux appels à `test_pythagore2`.

[15]:
```
def test_pythagore3(A, l, theta, phi):
    plot_carre(A, l, theta, 'k')
    u, v = base(theta)
    l1 = l * math.cos(phi)
    A1 = A + l * v
    test_pythagore2(A1, l1, theta + phi, phi)
    u, v = base(theta + phi)
    A2 = A1 + l1 * u
    l2 = l * math.sin(phi)
    test_pythagore2(A2, l2, theta + phi - math.pi / 2, phi)
```

[16]:
```
test_pythagore3(Vecteur(1, 1), 1, 0, math.pi / 6)
plt.axis('equal')
plt.grid()
```



2.1.2 2.2 Récursion, premier essai

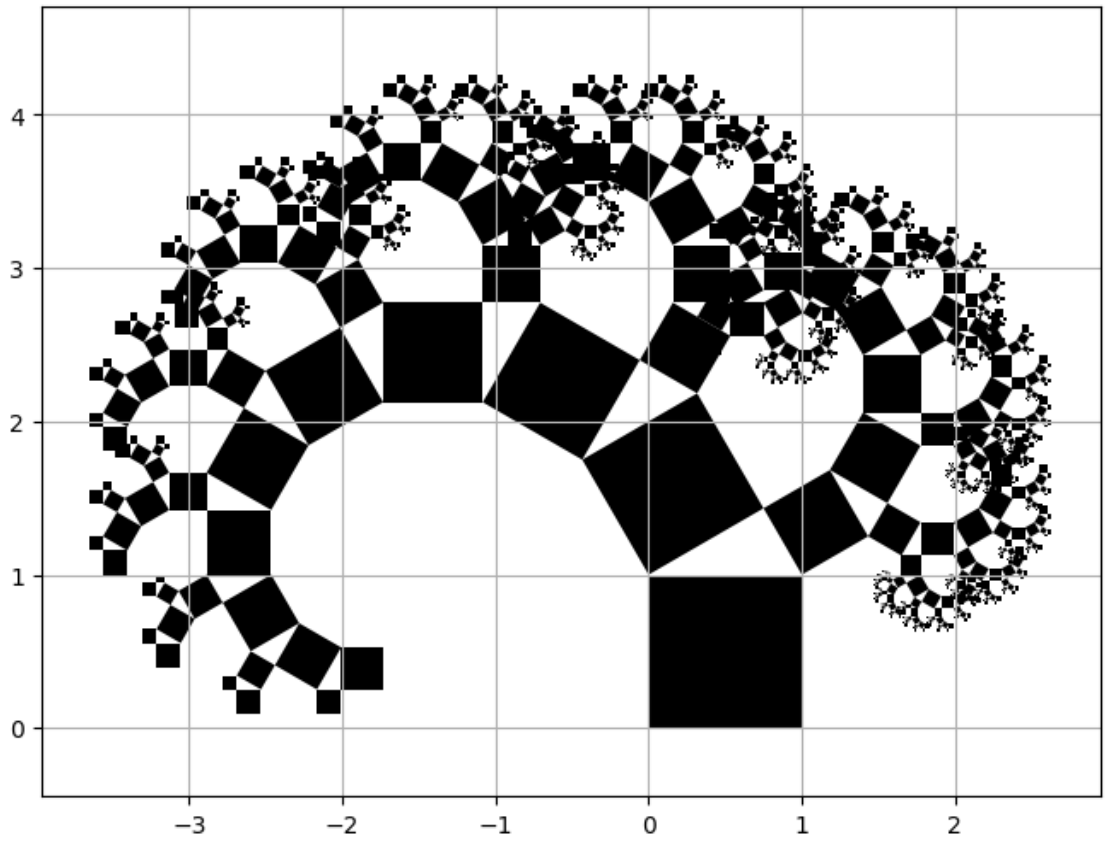
Au lieu d'écrire un fonction `test_pythagore10` qui appelle `test_pythagore9`… qui appelle `test_pythagore`, écrivons une fonction `pythagore0` qui prend un entier n en paramètre. La fonction se rappelle deux fois avec $n-1$ à la place de n. On obtient le code suivant.

[17]:
```
def pythagore0(A, l, theta, phi, n):
    plot_carre(A, l, theta, 'k')
    if n > 0:
        u, v = base(theta)
        l1 = l * math.cos(phi)
        A1 = A + l * v
        pythagore0(A1, l1, theta + phi, phi, n - 1)
        u, v = base(theta + phi)
        A2 = A1 + l1 * u
        l2 = l * math.sin(phi)
        pythagore0(A2, l2, theta + phi - math.pi / 2, phi, n - 1)
```

[18]:
```
cptr.reset()
pythagore0(Vecteur(0, 0), 1, 0, math.pi / 6, 9)
```

```
plt.axis('equal')
plt.grid()
print(cptr.val)
```

```
1023
```



Remarquons l'utilisation du compteur `cptr`. Celui-ci a été incrémenté 1023 fois : il y a 1023 carrés sur la figure.

Nous obtenons déjà quelque chose d'intéressant, mais de pas tout à fait abouti. Les branches sur la droite de l'arbre ont l'air plus « complètes » que celles de gauche. On pourrait évidemment augmenter la valeur de n pour obtenir un dessin plus joli, mais soyons bien conscients que le nombre de carrés tracés augmente exponentiellement en fonction de n. En effet, notons $C(n)$ le nombre de carrés tracés lors de l'appel `pythagore0(A, l, theta, phi, n)`.

On a $C(0) = 0$ et, pout tout $n \geq 1$, $C(n+1) = 2C(n) + 1$. On en déduit facilement par récurrence sur n que pour tout $n \in \mathbb{N}$,

$$C(n) = 2^{n+1} - 1$$

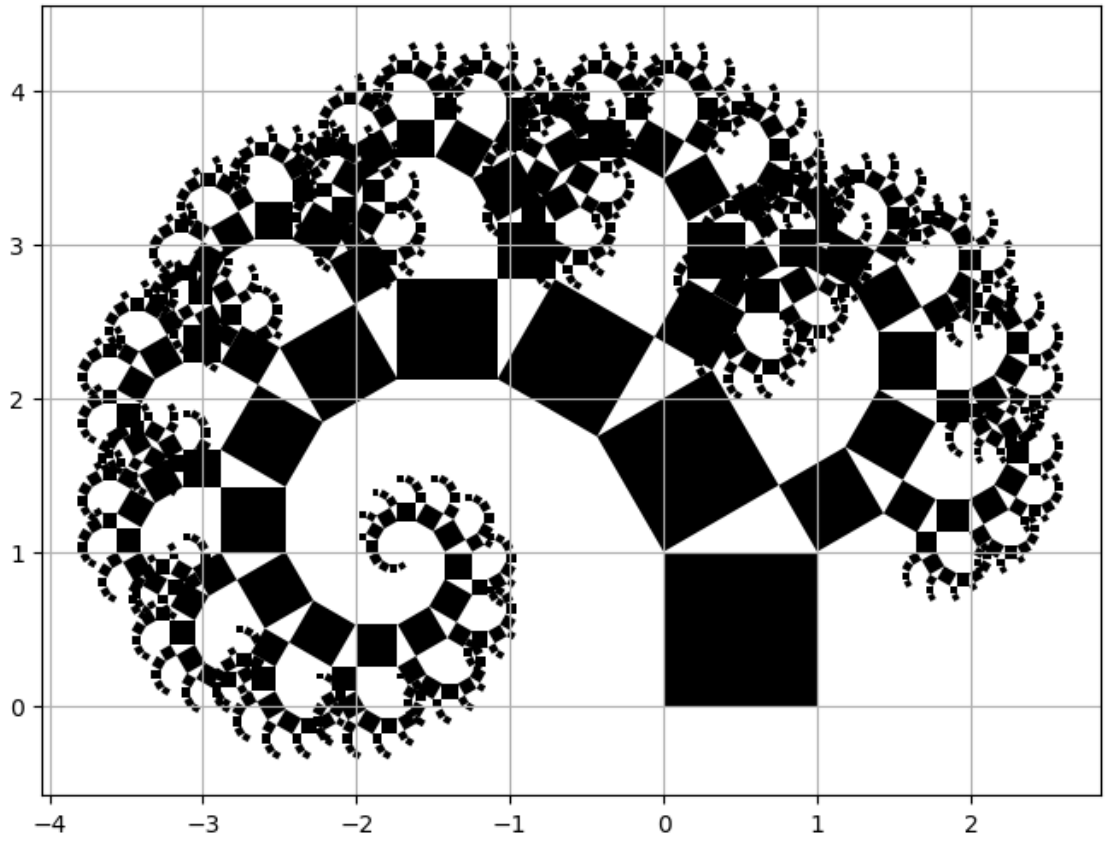
2.1.3 2.3 Une meilleure condition d'arrêt

Au lieu de faire terminer la fonction lorsque le paramètre n est nul, faison plutôt terminer celle-ci lorsque la longueur des côtés du carré à tracer est plus petite qu'une certaine valeur ℓ_{min}. Nous obtenons le code suivant.

[19]:
```
def pythagore1(A, l, theta, phi, lmin):
    if l >= lmin:
        plot_carre(A, l, theta, 'k')
        u, v = base(theta)
        l1 = l * math.cos(phi)
        A1 = A + l * v
        pythagore1(A1, l1, theta + phi, phi, lmin)
        u, v = base(theta + phi)
        A2 = A1 + l1 * u
        l2 = l * math.sin(phi)
        pythagore1(A2, l2, theta + phi - math.pi / 2, phi, lmin)
```

[20]:
```
cptr.reset()
pythagore1(Vecteur(0, 0), 1, 0, math.pi / 6, 0.04)
plt.axis('equal')
plt.grid()
print(cptr.val)
```

```
1035
```



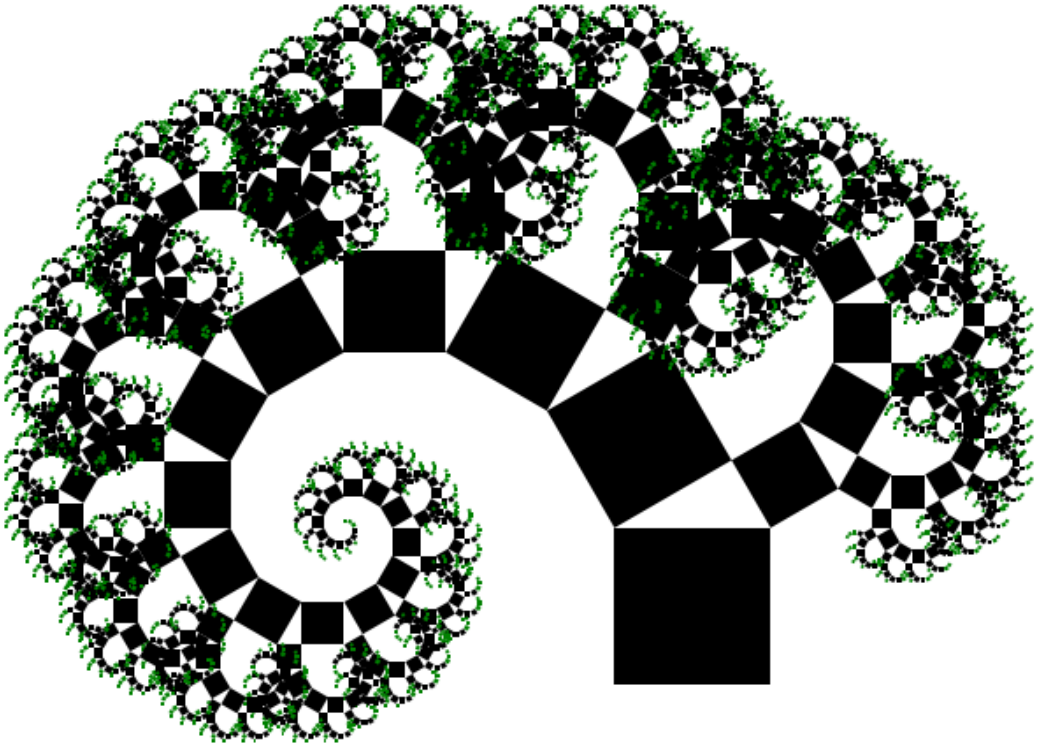
C'est déjà beaucoup mieux. Comme un arbre se doit de posséder des feuilles vertes, améliorons encore un tout petit peu notre code.

[21]:
```
def pythagore2(A, l, theta, phi, lmin):
    if l >= lmin:
        if l >= 1.5 * lmin: plot_carre(A, l, theta, 'k')
        else: plot_carre(A, l, theta, 'g')
        u, v = base(theta)
        l1 = l * math.cos(phi)
        A1 = A + l * v
        pythagore2(A1, l1, theta + phi, phi, lmin)
        u, v = base(theta + phi)
        A2 = A1 + l1 * u
        l2 = l * math.sin(phi)
        pythagore2(A2, l2, theta + phi - math.pi / 2, phi, lmin)
```

[24]:
```
cptr.reset()
pythagore2(Vecteur(0, 0), 1, 0, math.pi / 6, 0.02)
plt.axis('equal')
plt.axis('off')
```

```
#plt.grid()
print(cptr.val)
```

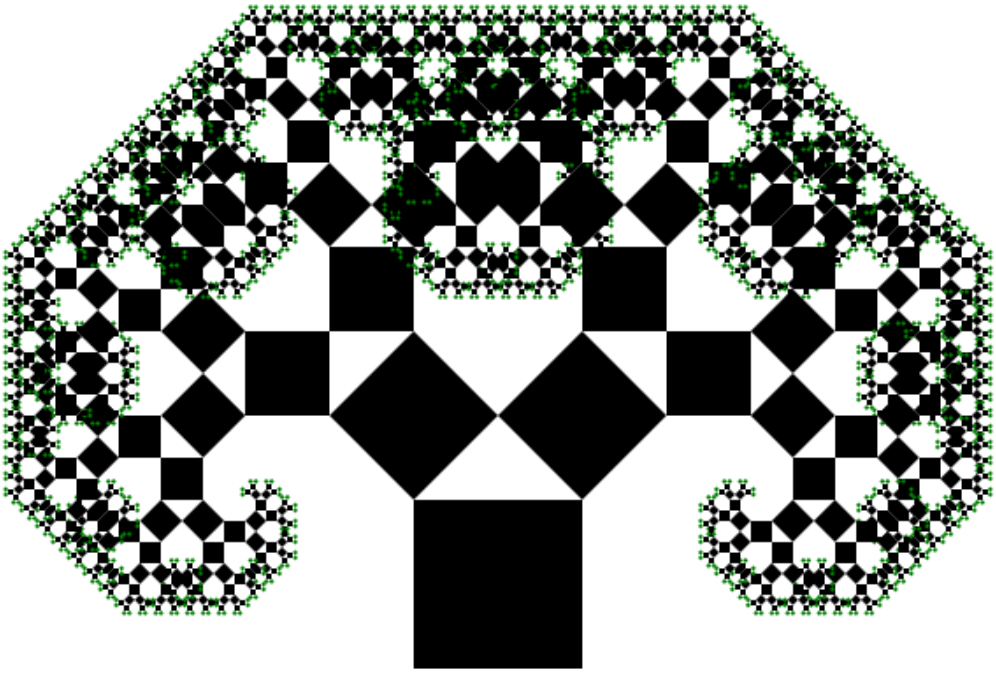
```
4140
```



C'est déjà beaucoup mieux. Voici ce que l'on obtient en prenant un angle de rotation égal à $\frac{\pi}{4}$.

[25]:
```
cptr.reset()
pythagore2(Vecteur(0, 0), 1, 0, math.pi / 4, 0.02)
plt.axis('equal')
plt.axis('off')
#plt.grid()
print(cptr.val)
```

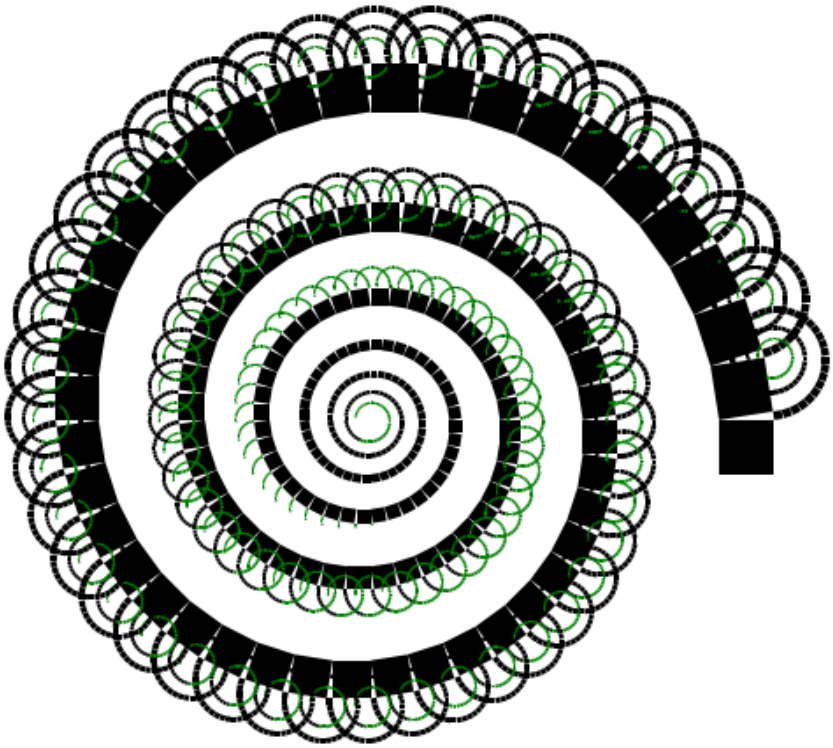
```
4095
```



Tentons également de prendre un angle presque égal à 0.

[26]:
```
cptr.reset()
pythagore2(Vecteur(0, 0), 1, 0, math.pi / 20, 0.04)
plt.axis('equal')
plt.axis('off')
#plt.grid()
print(cptr.val)
```

```
6476
```



2.1.4 2.4 Combien de carrés ?

Notons ℓ la longueur du carré initial et ℓ_0 la longueur minimale des carrés tracés.

- Au niveau 0 de l'arbre, il y a un carré de côté ℓ.
- Au niveau 1 de l'arbre, il y a deux carrés de côtés $\ell\cos\varphi$ et $\ell\sin\varphi$.
- Le carré de côté $\ell\cos\varphi$ donne naissance, au niveau 2 de l'arbre, à deux carrés de côtés $\ell\cos^2\varphi$ et $\ell\cos\varphi\sin\varphi$. De même, le carré de côté $\ell\sin\varphi$ donne naissance, au niveau 2 de l'arbre, à deux carrés de côtés $\ell\sin\varphi\cos\varphi$ et $\ell\sin^2\varphi$.

Soit $n \in \mathbb{N}$. Une récurrence facile sur n montre qu'au niveau n de l'arbre on a 2^n carrés. Les longueurs des côtés des carrés du niveau n sont de la forme $\ell\cos^k\varphi\sin^{n-k}\varphi$. où $0 \leq k \leq n$. Plus précisément, pour tout $k \in [|0, n|]$, il y a $\binom{n}{k}$ carrés de côté $\ell\cos^k\varphi\sin^{n-k}\varphi$. Un tel carré est tracé si et seulement si

$$\ell\cos^k\varphi\sin^{n-k}\varphi \geq \ell_0$$

Passons au logarithme pour obtenir

$$k\ln\cot\varphi \geq \ln\ell_0 - \ln\ell - n\ln\sin\varphi$$

2.4.1 Un cas particulier Prenons tout d'abord $\varphi = \frac{\pi}{4}$. On a dans ce cas $\ln\cot\varphi = 0$. L'inégalité devient

$$n \ln\sin\frac{\pi}{4} \geq \ln\ell_0 - \ln\ell$$

c'est à dire

$$n \leq \frac{\ln\ell_0 - \ln\ell}{\ln\sin\frac{\pi}{4}}$$

Posons

$$N = \left\lfloor \frac{\ln\ell_0 - \ln\ell}{\ln\sin\frac{\pi}{4}} \right\rfloor$$

Le nombre de carrés tracés est

$$\sum_{n=0}^{N} 2^n = 2^{N+1} - 1$$

[27]:
```
def nombre_carres1(l, lmin):
    N = math.floor((math.log(lmin) - math.log(l)) / math.log(math.sin(math.pi /␣
 ↪4)))
    return 2 ** (N + 1) - 1
```

[29]:
```
nombre_carres1(1, 0.02)
```

[29]: 4095

2.4.2 Le cas $\varphi \neq \frac{\pi}{4}$ Supposons maintenant $\varphi \neq \frac{\pi}{4}$. Remarquons que le nombre carrés tracés pour φ est le même que le nombre de carrés tracés pour $\frac{\pi}{2} - \varphi$. Nous pouvons donc prendre $0 < \varphi < \frac{\pi}{4}$, de sorte que $\ln\cot\varphi > 0$. On obtient dans ce cas

$$k \geq \frac{\ln\ell_0 - \ln\ell - n\ln\sin\varphi}{\ln\cot\varphi}$$

On a de plus $0 \leq k \leq n$. Pour qu'il y ait des carrés tracés au niveau n de l'arbre, il faut donc

$$\frac{\ln\ell_0 - \ln\ell - n\ln\sin\varphi}{\ln\cot\varphi} \leq n$$

ou encore

$$\ln\ell_0 - \ln\ell \leq n(\ln\cot\varphi + \ln\sin\varphi)$$

Remarquons que

$$\ln\cot\varphi + \ln\sin\varphi = \ln(\cot\varphi\sin\varphi) = \ln\cos\varphi$$

Notre condition sur n s'écrit donc

$$\ln\ell_0 - \ln\ell \leq n\ln\cos\varphi$$

ou encore, comme $\ln\cos\varphi < 0$ et n est entier,

$$n \leq \left\lfloor \frac{\ln\ell_0 - \ln\ell}{\ln\cos\varphi} \right\rfloor$$

La fonction `nmaxi` renvoie le membre de droite de l'inégalité ci-dessus.

[30]:
```
def nmaxi(phi, l, lmin):
    return math.floor((math.log(lmin) - math.log(l)) / math.log(math.cos(phi)))
```

Par exemple, pour $\varphi = \frac{\pi}{6}$, $\ell = 1$ et $\ell_0 = 0.02$, on dessine des carrés jusqu'au niveau 27 de l'arbre.

[31]:
```
nmaxi(math.pi / 6, 1, 0.02)
```

[31]: 27

La fonction `kmini` renvoie, pour un n donné, le plus petit entier $k \in [|0, n|]$ pour lequel on trace un carré de côté $\ell\cos^k\varphi\sin^{n-k}\varphi$.

[32]:
```
def kmini(phi, l, lmin, n):
    c, s = math.cos(phi), math.sin(phi)
    k = math.ceil((math.log(lmin) - math.log(l) - n * math.log(s)) / math.log(c /
 ↪ s))
    if k < 0: k = 0
    return k
```

[33]:
```
for n in range(nmaxi(math.pi / 6, 1, 0.02) + 1):
    print(n, kmini(math.pi / 6, 1, 0.02, n))
```

```
0 0
1 0
2 0
3 0
4 0
5 0
6 1
7 2
8 3
9 5
```

```
10 6
11 7
12 9
13 10
14 11
15 12
16 14
17 15
18 16
19 17
20 19
21 20
22 21
23 22
24 24
25 25
26 26
27 27
```

La fonction `power_dn` prend en paramètres deux entiers n et k. Elle renvoie

$$n^{\underline{k}} = \prod_{j=0}^{k-1} (n-j)$$

Remarquons que $k^{\underline{k}} = k!$.

[34]:
```python
def power_dn(n, k):
    p = 1
    for j in range(k): p *= (n - j)
    return p
```

La fonction `binom` prend en paramètres deux entiers n et k. Elle renvoie

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!}$$

On convient que $\binom{n}{k} = 0$ si $k < 0$.

[35]:
```python
def binom(n, k):
    if k < 0: return 0
    else:
        return power_dn(n, k) // power_dn(k, k)
```

[36]:
```python
print([binom(6, k) for k in range(-3, 10)])
```

```
[0, 0, 0, 1, 6, 15, 20, 15, 6, 1, 0, 0, 0]
```

Enfin, la fonction `nombre_carres2` renvoie le nombre de carrés tracés par la fonction `pythagore2`.

[37]:
```
def nombre_carres2(phi, l, lmin):
    if phi == math.pi / 4: return nombre_carres1(l, lmin)
    elif phi > math.pi / 4: phi = math.pi / 2 - phi
    s = 0
    for n in range(nmaxi(phi, l, lmin) + 1):
        for k in range(kmini(phi, l, lmin, n), n + 1):
            s += binom(n, k)
    return s
```

Testons.

[38]:
```
nombre_carres2(math.pi / 6, 1, 0.02)
```

[38]: 4140

[39]:
```
nombre_carres2(math.pi / 20, 1, 0.04)
```

[39]: 6476

Nous retrouvons bien les nombres qui étaient contenus dans le compteur `cptr`.

2.1.5 2.5 L'aire de l'arbre ?

Soit C un carré du niveau n de l'arbre de Pythagore, de côté a. Soient C' et C'' les deux fils de C, de côtés respectifs b et c. On a $a^2 = b^2 + c^2$, donc l'aire de C est égale à la somme des aires de C' et de C''. Ainsi, la somme des aires des carrés du niveau n de l'arbre ne dépend pas de n. Cette somme est donc égale à l'aire ℓ^2 du carré de départ. La somme des aires des carrés des niveaux 0, 1, ..., N est donc $(N+1)\ell^2$. Lorsque N tend vers l'infini, cette quantité tend vers $+\infty$.

L'arbre de Pythagore aurait-il une aire infinie ? Cet arbre est pourtant une partie bornée du plan ? En fait, non. Les carrés qui forment l'arbre ne sont pas disjoints, l'aire de l'arbre n'est donc pas la somme des des aires des carrés.

2.2 3. Des arbres, encore

2.2.1 3.1 Introduction

Nous allons donner ici quelques possibilités pour modifier la forme des arbres de Pythagore.

Les arbres que nous avons dessinés jusqu'à présent ressemblent plutôt à des buissons. Les branches ont tendance à s'enrouler, ceci d'autant plus vite que l'angle de rotation φ est proche de 0 ou de $\frac{\pi}{2}$. Pour éviter ces enroulements, l'idée est de changer la valeur de φ à chaque appel récursif.

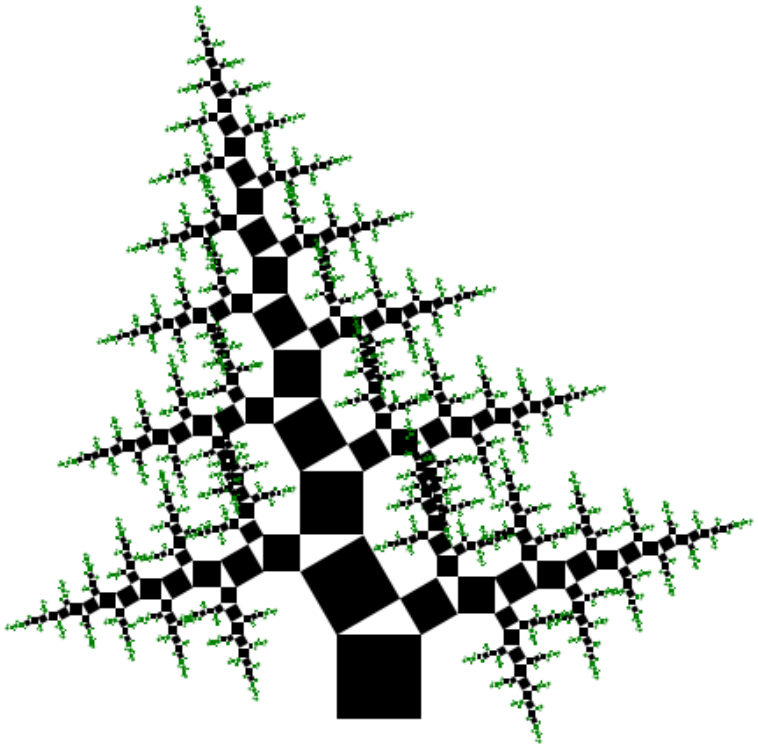
2.2.2 3.2 Alterner les angles.

Commençons par une fonction `pythagore3`qui, à chaque appel récursif, remplace φ par $\frac{\pi}{2} - \varphi$. Nous devrions cette fois obtenir des arbres droits.

[40]:
```
def pythagore3(A, l, theta, phi, lmin):
    if l > 1.5 * lmin: plot_carre(A, l, theta, 'k')
    else: plot_carre(A, l, theta, 'g')
    if l > lmin:
        u, v = base(theta)
        l1 = l * math.cos(phi)
        A1 = A + l * v
        pythagore3(A1, l1, theta + phi, math.pi / 2 - phi, lmin)
        u, v = base(theta + phi)
        A2 = A1 + l1 * u
        l2 = l * math.sin(phi)
        pythagore3(A2, l2, theta + phi - math.pi / 2, math.pi / 2 - phi, lmin)
```

[41]:
```
pythagore3(Vecteur(0, 0), 1, 0, math.pi / 6, 0.04)
plt.axis('equal')
plt.axis('off')
#plt.grid()
```

[41]: (-4.373219714321845, 5.373219714321843, -0.7203031961397102, 8.892391982221708)



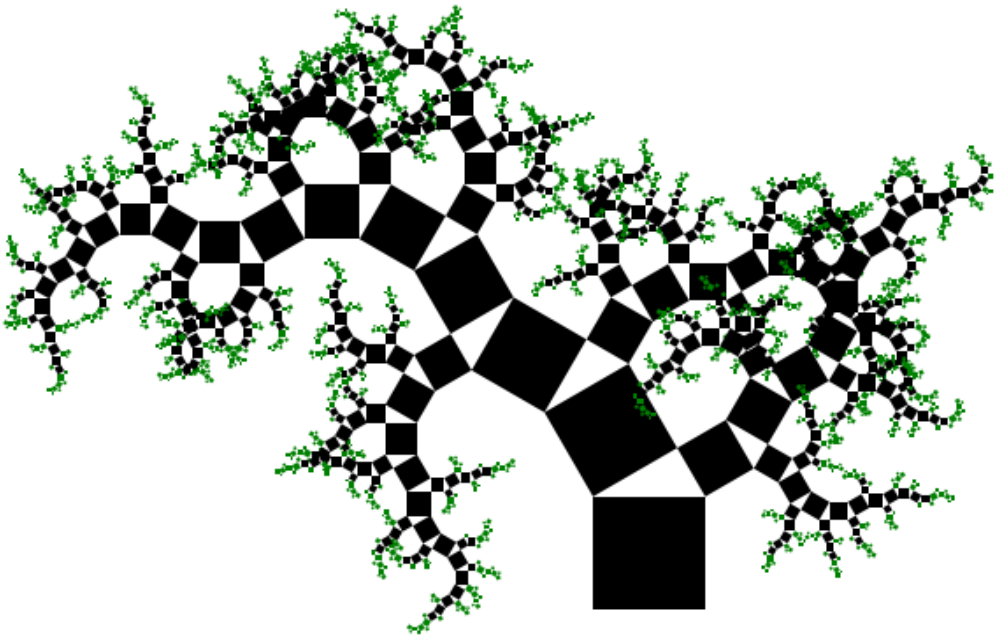
2.2.3 3.2 Alterner aléatoirement les angles.

Au lieu d'alterner les angles à chaque appel récursif, alternons-les aléatoirement.

[42]:
```
def pythagore4(A, l, theta, phi, lmin, t):
    if l > 1.5 * lmin: plot_carre(A, l, theta, 'k')
    else: plot_carre(A, l, theta, 'g')
    if l > lmin:
        if random.uniform(0, 1) < t: phi1 = phi
        else: phi1 = math.pi / 2 - phi
        u, v = base(theta)
        l1 = l * math.cos(phi)
        A1 = A + l * v
        pythagore4(A1, l1, theta + phi, phi1, lmin, t)
        u, v = base(theta + phi)
        A2 = A1 + l1 * u
        l2 = l * math.sin(phi)
        pythagore4(A2, l2, theta + phi - math.pi / 2, phi1, lmin, t)
```

[43]:
```
pythagore4(Vecteur(0, 0), 1, 0, math.pi / 6, 0.04, 0.5)
plt.axis('equal')
plt.axis('off')
#plt.grid()
```

[43]:
```
(-5.698564704397139,
 4.004732105517407,
 -0.5049687756909809,
 5.6593912779214035)
```



2.2.4 3.3 Encore une possibilité

Généralisons ce que nous avons fait au paragraphe 3.1. Au lieu d'alterner entre l'angle φ et l'angle $\frac{\pi}{2} - \varphi$, alternons entre plusieurs angles $\varphi_1, \dots, \varphi_m$. Lorsque la fonction est appelée avec la liste $[\varphi_1, \dots, \varphi_m]$, elle utilise le théorème de Pythagore avec l'angle φ_1, puis se rappelle récursivement avec la liste $[\varphi_m, \varphi_1, \dots, \varphi_{m-1}]$.

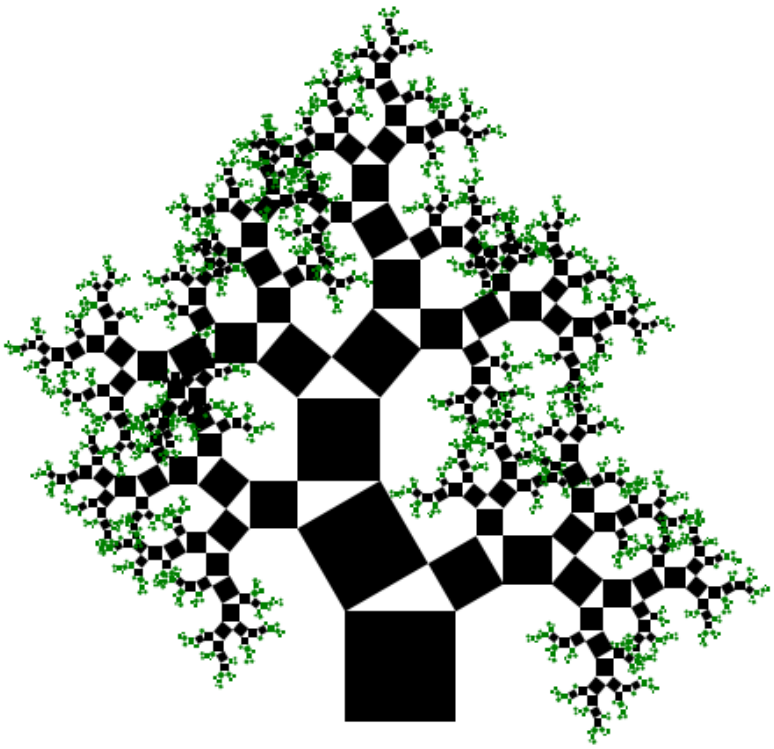
[44]:
```
def pythagore5(A, l, theta, phis, lmin):
    if l > 1.5 * lmin: plot_carre(A, l, theta, 'k')
    else: plot_carre(A, l, theta, 'g')
    if l > lmin:
        u, v = base(theta)
        phi = phis[0]
        l1 = l * math.cos(phi)
        A1 = A + l * v
        phis = [phis[-1]] + phis[:-1]
        pythagore5(A1, l1, theta + phi, phis, lmin)
        u, v = base(theta + phi)
        A2 = A1 + l1 * u
        l2 = l * math.sin(phi)
```

```
        pythagore5(A2, l2, theta + phi - math.pi / 2, phis, lmin)
```

[45]:
```
def deg(d): return d * math.pi / 180
```

[46]:
```
pythagore5(Vecteur(0, 0), 1, 0, [deg(30), deg(40), deg(50), deg(60)], 0.04)
plt.axis('equal')
plt.axis('off')
#plt.grid()
```

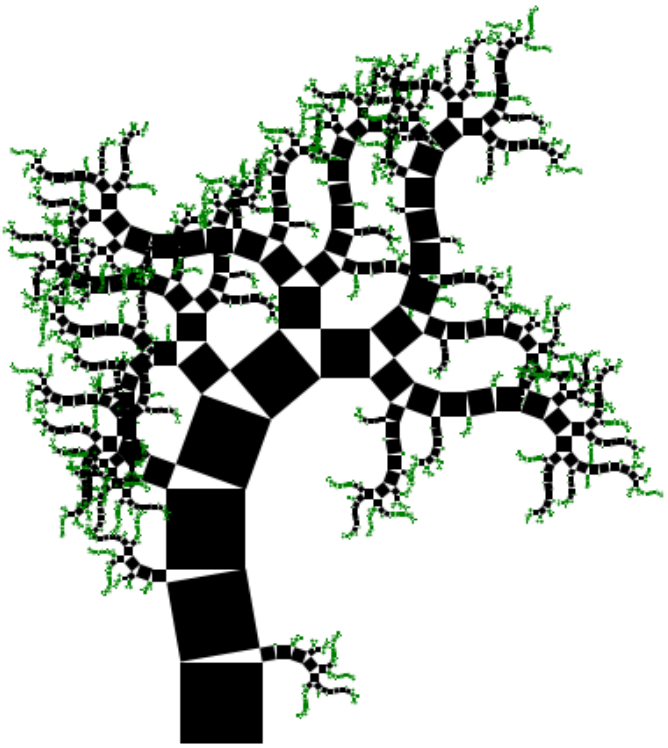
[46]: (-3.4388461761307836, 4.194160366888336, -0.5361996016904149, 6.80472202630163)



[47]:
```
pythagore5(Vecteur(0, 0), 1, 0, [deg(10), deg(20), deg(30), deg(40), deg(50),␣
 ↪deg(60), deg(70), deg(80)], 0.04)
plt.axis('equal')
plt.axis('off')
#plt.grid()
```

[47]: (-2.588469941135587,
6.322367592113404,
-0.45194116539536466,

```
9.490764473302658)
```



On laisse le lecteur s'amuser en changeant les valeurs des angles ...