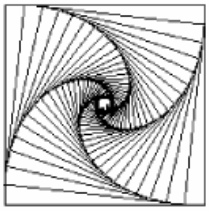
Carrés imbriqués



On souhaite programmer en Python une figure particulière, celle de carrés imbriqués les uns dans les autres. On utilisera pour cela le module Turtle, module graphique permettant de piloter une tortue à l'écran à l'aide d'instructions basiques.
Il faudra donc, en début de script, ajouter une commande `from turtle import *` afin de pouvoir utiliser les instructions de Turtle.

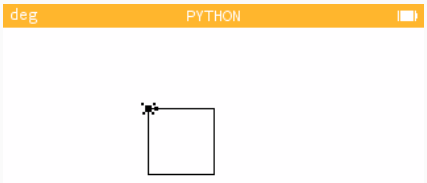
Dessiner un carré

1. A l'exécution du script, la tortue est positionnée au centre de l'écran et dirigée vers la droite.
   A l'aide des instructions `forward(x)`, qui permet de faire avancer la tortue de x pixels (l'écran de la calculatrice fait 320 pixels en longueur et 222 pixels en hauteur), et `right(a)`, qui permet de faire pivoter la tortue de a degrés sur sa droite, écrire une fonction `carre(l)` permettant de dessiner un carré de côté l de façon à ce que la tortue reprenne en fin de programme sa position initiale, tournée vers la droite.
   Avec quelle instruction peut-on limiter le nombre de lignes de code ?

   L'utilisation d'une boucle `for i in range(4)` permettra de limiter les lignes de code.

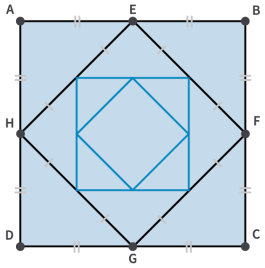
2. Il est possible de déplacer la tortue ailleurs sur l'écran à l'aide de l'instruction `goto(x,y)` permettant de déplacer la tortue vers le point de coordonnées (x;y), le point (0;0) correspondant au centre de l'écran. Cependant, tout déplacement de la tortue laisse une trace si le "stylo" qu'elle tient en main n'est pas relevé. A l'aide des instructions `penup()` et `pendown()`, écrire une nouvelle fonction `essai()` permettant de tracer un carré de côté 50px au point de coordonnées (-50 ; 50).

```
deg                PYTHON
1 from math import *
2 from turtle import *
3
4 def carre(l):
5     for i in range(4):
6         forward(l)
7         right(90)
8
9 def essai():
10    penup()
11    goto(-50,50)
12    pendown()
13    carre(50)
14
15
16
```



Carrés imbriqués - niveau 1

On souhaite imbriquer les carrés comme ci-contre, c'est-à-dire de façon à ce que les sommets du carré intérieur soient situés au milieu des côtés du carré supérieur.



1. Après l'exécution de la fonction initiale, permettant de tracer le carré ABCD de côté l, comment amener la tortue en position E (voir la figure) et la pivoter pour tracer le second carré EFGH ?

   La tortue trace son second carré à partir du milieu du côté du premier carré. Etant donné sa position et son orientation à l'issue du premier dessin, il suffit d'utiliser l'instruction `forward(l/2)` pour qu'elle se positionne à la moitié du segment.
   Il faut maintenant calculer l'angle de pivot $\widehat{BEF}$ afin de donner la bonne orientation à notre tortue. Le triangle BEF est isocèle rectangle en B d'où un angle de 45° que l'on code avec l'instruction `right(45)`.

2. Calculer la longueur du côté du nouveau carré.

   En utilisant le théorème de Pythagore dans le triangle BEF, on obtient $\sqrt{\frac{l^2}{2}}$.

3. Compléter le programme suivant permettant d'obtenir la figure avec n carrés et l'exécuter sur la calculatrice.

```
from math import *
from turtle import *

def carre(l):
  for i in range(4):
    forward(l)
    right(90)

def figure(... , ...):
  penup()
  goto(... , ...)
  pendown()
  for i in range(n):
    carre(l)
    forward(... , ...)
    right(...)
    l= ...
```

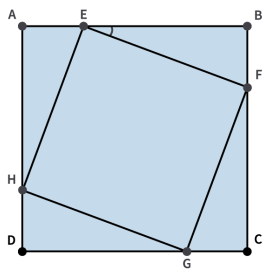
```
from math import *
from turtle import *

def carre(l):
  for i in range(4):
    forward(l)
    right(90)

def figure(l,n):
  penup()
  goto(-50,50)
   pendown()
  for i in range(n):
    carre(l)
    forward(l*0.5)
    right(45)
    l=sqrt((l**2)/2)
```

Carrés imbriqués - niveau 2

On souhaite maintenant réaliser une nouvelle fonction `figure2(l,n)` identique à la précédente mais en changeant les conditions de construction.
Le sommet du second carré doit maintenant se trouver à un quart du côté du carré précédent. Autrement dit, si l'on reprend la figure précédente : $AE = \frac{1}{4}AB$.



1. A l'aide des relations trigonométriques, exprimer l'angle $\widehat{BEF}$ en fonction de l, longueur de AB.

   Le triangle BEF est rectangle en B. D'après les relations trigonométriques, on obtient :

   $$\widehat{BEF} = \tan^{-1}\left(\frac{BF}{EB}\right) = \tan^{-1}\left(\frac{0.25\,l}{0.75\,l}\right) = \tan^{-1}\left(\frac{1}{3}\right)$$

2. La fonction `atan()` du module `math` renvoie un angle en radians tandis que les changements d'orientation du module `turtle` sont exprimés en degrés. Quelle ligne de code faudrait-il écrire pour calculer l'angle $\widehat{BEF}$ à partir de la relation précédente puis le transformer en degrés ?

   ```
   angle =atan(1/3)*180/pi
   ```

3. Exprimer maintenant la longueur EF en fonction de l.

Le triangle BEF est rectangle en B. D'après le théorème de Pythagore, on obtient :

$$EF = \sqrt{(0.75\,l)^2 + (0.25\,l)^2} = \frac{l}{4}\sqrt{10}$$

4. En réutilisant le programme du niveau 1, écrire une fonction `figure2(l,n)` permettant de tracer n carrés imbriqués dans un carré de longueur l en respectant les conditions de construction.

On appelle à nouveau la fonction `carre(l)` définie au tout début du script.

```
def figure2(l,n):
  penup()
  goto(-50,50)
  pendown()
  for i in range(n):
    carre(l)
    forward(l*0.25)
    angle=atan(1/3)*180/pi
    right(angle)
    l=sqrt((0.75*l)**2+(0.25*l)**2)
```

Carrés imbriqués - dernier niveau

On aimerait maintenant modifier notre programme pour que les carrés imbriqués soient très resserrés, tout comme sur la première illustration de cette activité.
On propose de créer une dernière fonction qui prendra en argument non seulement l longueur du premier carré, n nombre de carrés, mais aussi a coefficient inférieur à 1 tel que $AE = a \times AB$.

1. En reprenant le même raisonnement qu'au-dessus, déterminer la valeur de l'angle $\widehat{BEF}$ en fonction de l.

$$\widehat{BEF} = \text{atan}\left(\frac{a}{1-a}\right)$$

2. Exprimer maintenant la longueur EF en fonction de l.

$$EF = \sqrt{((1-a)\ l)^2 + (al)^2}$$

3. Ecrire une fonction `figure3(l,n,a)` permettant de tracer n carrés imbriqués dans un carré de longueur l en respectant les conditions de construction.

```
def figure3(l,n,a):
  penup()
  goto(-50,50)
  pendown()
  for i in range(n):
    carre(l)
    forward(l*a)
    angle=atan(a/(1-a))*180/pi
    right(angle)
    l=sqrt(((l-a)*l)**2+(a*l)**2)
```

On peut proposer aux élèves plus avancés de trouver des variantes en utilisant, par exemple, un autre type de boucle : le programme dessine des carrés jusqu'à ce que la longueur l soit inférieure à 5 px. On peut aussi proposer aux élèves de positionner le carré sur l'écran en fonction de la valeur de l, d'ajouter des couleurs, etc.
Voir la fonction `variante(l,a)` du script :

```
def variante(l,a):
  penup()
  x=-160+((320-l)/2)
  y=111-((222-l)/2)
  goto(x,y)
  pendown()
  color(0,0,255)
  while l>5:
    color(l,255-l,255-l)
    carre(l)
    forward(l*a)
    angle=atan(a/(1-a))*180/pi
    right(angle)
    l=sqrt(((1-a)*l)**2+(a*l)**2)
```

4. Quelle ligne de code pourrait-on ajouter pour renvoyer un message d'erreur si l'utilisateur entre une valeur de a supérieure à 1 ?

Il suffit d'ajouter `if a>1: return "Impossible"` en début de programme.